НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО» ІНСТИТУТ ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ
КАФЕДРА МАТЕМАТИЧНИХ МЕТОДІВ СИСТЕМНОГО АНАЛІЗУ

На правах рукопису
УДК 004.93`1

До захисту допущено
Завідувач кафедри
_____ О.Л Тимощук
(підпис) (ініціали, прізвище)
«____»_____2020 р.

Магістерська дисертація

на здобуття ступеня магістра за спеціальністю 122 «Комп'ютерні науки»
на тему*: «*Система двовимірного розпізнавання об'єктів в безпілотних автомобілях»

Виконав:
студент ІІ курсу, групи КА-93мп _____
Яковенко Сергій Сергійович

Керівник:
доцент кафедри ММСА _____
д.т.н. доц. Недашківська Н.І.

Рецензент:
доцент кафедри системного проектування _____
КПІ ім. Ігоря Сікорського
к.т.н. доц. Безносик О.Ю.

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань

Студент _____

Київ
2020

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО» ІНСТИТУТ ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ
КАФЕДРА МАТЕМАТИЧНИХ МЕТОДІВ СИСТЕМНОГО АНАЛІЗУ

Рівень вищої освіти — другий (магістерський)
Спеціальність (спеціалізація) — 122 «Комп'ютерні науки» («Системи штучного інтелекту»)

ЗАТВЕРДЖУЮ
В. о. завідувача кафедри ММСА
О. Л. Тимощук
«___» _____ 2020 р.

ЗАВДАННЯ
на магістерську дисертацію студенту Яковенко Сергію Сергійовичу

1. **Тема дисертації:** «Система двовимірного розпізнавання об'єктів в безпілотних автомобілях», науковий керівник дисертації _доцент кафедри ММСА к.т.н. Недашківська Н.І.
затверджені наказом по університету від 02.11.2020 № 3182-с

2. **Термін подання студентом дисертації:** 14 грудня 2020 р.

3. **Об'єкт дослідження:** глибинні нейронні мережі

4**. Предмет дослідження:** застосування глибинних нейронних мереж в задачах розпізнавання образів

5. **Перелік завдань, які потрібно розробити:**
1) провести літературний огляд;
2) виконати порівняння наявних мереж, що використовуються для розпізнавання образів;
3) обрати підходящі параметри системи та провести навчання;
4) знайти оптимальні параметри системи;
5) реалізувати консольний додаток та провести його тестування.

6. **Орієнтовний перелік ілюстративного матеріалу:**
1) Презентація 10-15 слайдів.

7. Дата видачі завдання 01 вересня 2020 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1	Проведення літературний огляд	05.09.2020–11.09. 2020	
2	Виконання порівняння наявних мереж, що використовуються для розпізнавання образів	12.09.2020–21.09. 2020	
3	Вибір підходящі параметри системи та провести навчання	22.09.2020–05.10. 2020	
4	Пошук оптимальні параметри системи	06.10.2020–08.10. 2020	
5	Реалізація консольний додаток та провести його тестування	09.10.2020–20.11. 2020	
6	Розробка стартап проекту	21.11.2020-24.11. 2020	
7	Оформлення звіту	19.11.2020–10.12. 2020	

Студент _____ С.С. Яковенко
(підпис)

Науковий керівник дисертації _____ Н.І Недашківська
(підпис)

РЕФЕРАТ

Магістерська дисертація виконана на 106 сторінках, містить 37 ілюстрацій, 22 таблиці 15 джерел та 1 додаток.

Актуальність. В зв’язку з розвитком сфери штучного інтелекту, безпілотного автотранспорту, а також систем безпеки в автомобілях, постає необхідність в розробці яка була б здатна проводити детекцію об’єктів навколо машини за допомогою різноманітних датчиків, зокрема відеокамер

Мета дослідження – використовуючи машинне навчання, розробити систему розпізнавання об’єктів в відеопотоці

Об’єкт дослідження: задача двовимірного розпізнавання учасників дорожнього руху: велосипедистів, пішоходів, автомобілів.

Предмет дослідження: методи і моделі глибинних нейронних мереж в задачах розпізнавання образів.

Методи дослідження, застосовані у даній роботі, базуються на методах машинного навчання та експертної оцінки.

Наукова новизна одержаних результатів — розроблено систему, що розпізнає учасників дорожнього руху: велосипедистів, пішоходів, автомобілі; Дана система може використовувати смартфон водія в якості обчислювального сервера, що дозволяє встановлювати її навіть на старих автомобілях

Вдосконалити систему можна додавши інші канали отримання данних та використовувати їх для поліпшення результатів розпізнавання в складних умовах.

ЗГОРТКОВІ НЕЙРОННІ МЕРЕЖІ, ГЛИБОКІ НЕЙРОННІ МЕРЕЖІ, МАШИННЕ НАВЧАННЯ, РОЗПІЗНАВАННЯ ОБ’ЄКТІВ, АВТОМОБІЛІ

ABSTRACT

Master's thesis performed at 106 pages, contains 37 illustrations, 22 tables, 15 sources, 1 supplement.

Actuality. Due to the development of artificial intelligence, unmanned vehicles, as well as safety systems in cars, there is a need to develop a device that would be able to detect objects around the machine using a variety of sensors, including video cameras

The aim of the study — is to use machine learning to develop a system for recognizing objects in a video stream

The object of research: the problem of two-dimensional recognition of road users: cyclists, pedestrians, cars.

The subject of research: methods and models of deep neural networks in image recognition problems.

The research methods used in this work are based on the methods of machine learning and expert evaluation.

Scientific novelty of the obtained results - a program has been developed that recognizes road users: cyclists, pedestrians, cars;

You can improve the system by adding other data acquisition channels and use them to improve recognition results in difficult conditions.

CONVOLVED NEURAL NETWORKS, DEEP NEURAL NETWORKS, MACHINE LEARNING, OBJECT RECOGNITION, CARS

ЗМІСТ

ВСТУП

Із зростанням рівня життя, що спостерігається в більшості країн світу, невпинно зростає і рівень автомобілізації, а в країнах з досить низьким розвитком і якістю громадського транспорту є досить високим не лише рівень володіння автомобільним транспортом, а й рівень користування ним. В зв’язку з цим неодмінно зростає рівень аварійності, рівень смертності та травматизму на дорогах. Саме з цих причин найбільші автомобільні виробники впроваджують нові та модернізують існуючі системи безпеки, які отримують інформацію з десятків датчиків та у критичних ситуаціях здатні реагувати швидше за людину.

З розвитком обчислювальної потужності комп'ютерів та розділів науки пов’язаних з інформаційними технологіями, все більшої популярності здобувають системи, що здатні повністю замінити водія за кермом, але через високі стандарти безпеки, на звичайних дорогах такі автомобілі все ще складають відносно невеликий відсоток.

Метою магістерської дисертації є дослідження методів та алгоритмів, які можуть бути використані для двовимірного розпізнавання об'єктів в безпілотних автомобілях, та побудова системи для підвищення безпеки автомобільного руху за рахунок створення системи розпізнавання об’єктів, яка працює на смартфоні водія.

Предметом дослідження є порівняння алгоритмів розпізнавання об’єктів. Для досягнення поставленої мети були сформульовані та вирішені наступні задачі:

1) огляд та аналіз існуючих систем розпізнавання об’єктів;
2) побудова моделей для порівняння якості роботи різних алгоритмів розпізнавання;
3) аналіз отриманих результатів;
4) визначення оптимального алгоритму для розпізнавання об’єктів;
5) написання програмної реалізації алгоритму та його аналіз.

1 ОГЛЯД ІСНУЮЧИХ РІШЕНЬ З РОЗПІЗНАВАННЯ ОБ'ЄКТІВ

1.1 Загальна характеристика

Сучасні автомобілебудівники при розробці нових моделей значну увагу приділяють покращенню їх безпекових характеристик. Останнім часом все більшу роль в автомобільних системах безпеки відіграють компоненти побудовані з використанням штучного інтелекту. Ці рішення спрямовані не лише на забезпечення безпеки водія та пасажирів транспортного засобу, а й пішохоідв та інших учасників дорожнього руху

Системи детекціїї пішоходів використовується для запобігання зіткнення із нимми. Для цього використовуються різноманітні датчики та оптичні канали отримання даних інформація з яких постійно аналізується та у випадку помічених пішоходів, що потенційно можуть потрапити під колеса автомобіля, автоматично знижують швидкість, що в свою чергу, при неможливості уникнути зіткнення, підвищує шанси пішоходів на виживання. Також, найсучасніші системи здатні перехоплювати керування для об’їзду перешкоди.

Перші системи виявлення пішоходів встановлювалися на автомобілі Volvo в 2010 році та реалізовувала наступні функції:

1) детекція пішоходів;
2) попередження про небезпеку
3) автоматичне гальмування.

Дана система додатково може включати в себе подушки безпеки для пішоходів або інші засоби зменшення травматизму, але найважливішим єтапом даної системи без якої неможлива реалізація решти функціоналу є саме виявлення об’єкту та його класифікація.

Найбільшими перешкодами при вирішенні цієї задачі є висока швидкість зміни відстані до об’єктів та я їх положення у кадрі, також природні умови здатні

впливати на датчики автомобіля, а також навіть за сприятливої погоди об’єкт може бути нерозпізнаванм на певному фоні.

На сьогоднішній день існує близько двадцяти різноманітних автомобільних систем з розпізнавання об’єктів, більшість з них є розробками великих автомобільних концернів. Кожна з цих систем має свої переваги та недоліки відносно інших, а також використовують різноманітні методи розпізнавання. Цуй фактор вказує на те, що поки що сфера розпізнавання об’єктів за оптичними данними лише починає свій розвиток і напрямок її подальшого розвитку однозначно не обрано.

Системи оптичного розпізнавання містять кілька важливих характеристик за якими їх можна порівнювати. Першою характеристикою є розміри робочої зони. Зазвичай, камери та лідари забезпечують кут огляду близько 80 градусів за напрямом руху автомобіля, а також глибину до 30 метрів. Збільшувати цей параметр не має сенсу, оскільки із збільшенням відстані до об’єктів зростає і кількість шумів, а також зменшуються розміри об’єкту для спостерігача, що погіршує якість розпізнавання.

Другою характеристикою є швидкодія системи. Оскільки нам необхідно попередити водія про можливі проблеми на дорозі, очевидно що необхідно забезпечити можливість роботи в реальному часі, більше того, час обробки кожного фрейму повинен бути мінімальним, щоб забезпечити надійність роботи в якомога ширшому діапазоні. Оскільки при досить невеликих швидкостях для сучасних автомобілів, об’єкти в кадрі можуть повністю оновлюватися менше ніж за секунду.

Для забезпечення швидкодії використовують два підходи. Перший з них полягає у використанні більш потужних обчислювальних платформ, але такий варіант не завжди можливий через більшу вартість потужного обладнання. Другий підхід полягає у використанні більш простих алгоритмів, але за такого підходу доводиться жертвувати якістю розпізнавання, що робить його також не універсальним.

Важливим чинником, що впливає на якість роботи є освітленість. Оскільки надто яскраве світло (наприклад якщо їхати в напрямку сонця вдень) або навпаки нестача освітлення (при русі вночі на неосвітлених ділянках дороги) здатні значно погіршувати якість даних, що отримуються за допомогою камер, сучасні автомобілі преміум сегменту оснащені додатковими сенсорами.

1.2 Класифікація засобів отримання даних

1.2.1 Камера

Даний тип сенсора є найбільш дешевим серед усіх, а тому й найбільш розповсюджений. Для розпізнавання об’єктів отриманих за допомогою камер зазвичай використовують різноманітні згорткові нейронні мережі. Головним недоліком подібних сенсорів є те, що вони найбільше залежать від погодних факторів і в туман стають майже безпорадними. Також в умовах недостатньої освітленості, через збільшення кількості шумів на зображенні, значно погіршується якість розпізнавання. Для того щоб компенсувати дані недоліки, зазвичай використовують допоміжні датчика, що розглянуті далі.

1.2.2 ІЧ камера

Інфрачервоні камери від звичайних камер відрізняються наявністю спеціальної інфрачервоної підсвітки, що є невидимою для людського ока, але в той самий час світло з довжиною хвилі в ІЧ діапазоні є видимим для камер, саме тому така підсвітка дозволяє бачити зображення об’єктів навіть при недостатньому освітленні. Цей принцип роботи схожий на той, який використовувався в перших приладах нічного бачення. Він є досить простим, а тому його реалізація є досить дешевою.

Дані камери відносно звичайних камер мають як свої переваги так і недоліки. Серед переваг, як вже й було зазначено можливість роботи при поганому освітленні, а отже збільшується час на прийняття рішення водієм. Але недоліком є те, що такі камери все ще не можуть працювати в умовах туману або опадів і через підсвітку можуть отримувати ще гірші зображення ніж звичайні камери, а також не надто висока контрастність отриманого зображення.

Частково цю проблему можливо вирішити за допомогою тепловізійних камер, що вловлюють теплове випромінювання від об’єктів, Такі камери не мають підсвітки і дозволяють на ще більшій відстані помітити наприклад тварину чи пішохода, але в плані розпізнавання неживих об’єктів вони не мають жодних переваг, а їх роздільна здатність значно нижче. До того ж вдень влітку тепловізор починає вловлювати тепло і від нагрітих неживих об’єктів.

1.2.3 Радар

Радар є одним із двох типів сенсорів, що роблять можливим отримання доволі точні дані незалежно від видимості на дорозі, забезпечуючи однакову якість даних як в туман, так і вночі. Також вони дозволяють отримати доволі точні про відстань до об’єкту, але його ціна вища за ціну камер, хоча все ж і нижча за тепловізор. Також, на відміну від камер, дані отримуються у форматі для якого необхідно використовувати спеціальні алгоритми.

Загалом використання радарів є досить перспективною технологією. Такі висновки можна зробити на основі того, що наприкінці 90-х військові через високу ціну на тепловізійні матриці розглядали можливість побудови систем, що будують тривімирну модель на основі даних про відбиття радіохвиль від різних об’єктів

Також радари коштують дешевше за лідари і є менш вразливими до зовнішніх чинників (зокрема подряпина захисного скла на лінзі лідару або пилова завіса здатні викривляти результати вимірів).

1.2.4 Лідар

Найбільш сучасною, але й найбільш дорогою технологією є лідари. Вони будують найточнішу модель оточення, за допомогою зондування простору лазером. Загалом принцип схожий з тим який використовується в радарах: заміряється час між надсиланням лазерного сигналу та отримання його приймачем. Сучасні системи генерують близько мільйону імпульсів щосекунди, чим ця кількість більша, тим більша точність та роздільна здатність отриманої моделі. Як і для радарів, дані отримуються в форматі, що потребує додаткової обробки, що впливає на швидкість та продуктивність.

В деяких роботах використовується алгоритм кластеризації на основі відстані. Тобто за допомогою відстані до кожної точки будується фінальна модель. Однак даний алгоритм є досить вимогливим до ресурсів, а тому його швидкість не завжди є достатньою. В останній час використовують моделювання за допомогою 3D-вокселів та дискримінаційного аналізу. Даний метод є більш точним, та більш надійним.

На відміну від радару, в туманну погоду, через викривлення лазерних променів краплинами води, дані стають викривленими та неточними, також лінза лідару не повинна містити подряпин, оскільки це також викривляє промінь. На данний момент, лідар є перспективною, але все ще не досить відточеною технологією, до того ж, як зазначено вище, сучасні процесори не здатні забезпечити побудову моделі високої роздільної здатності.

Саме тому найчастіше лідар використовують в комбінації з різноманітними камерами, де на кадрі з камери розпізнаються потенційно

небезпечні об’єкти, а за допомогою лідару оцінюється його позиція відносно автомобілю. Даний підхід дозволяє знизити кількість точок для яких проводиться обчислення, а відповідно продуктивність системи зростає.

Приклад трьохвимірної моделі, отриманої за допомогою лідара показано на рисунку 1.1.

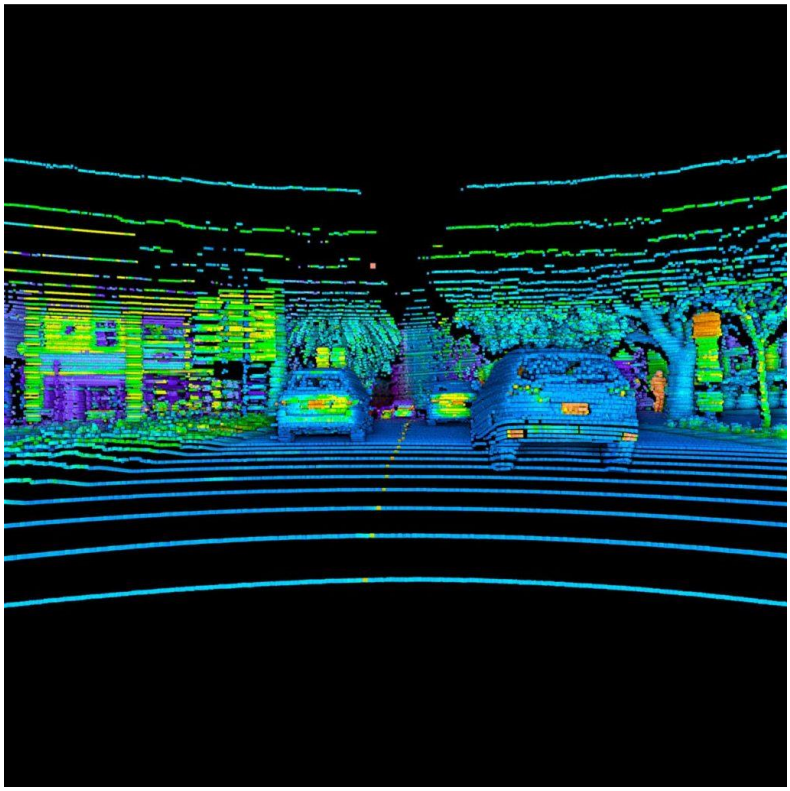

Рисунок 1.1 - Приклад моделі на основі даних лідару

1.3 Огляд наявних на ринку систем

1.3.1 PRE-SAFE Mercedes-Benz

В своїй основі PRE-SAFE® Mercedes-Benz - це система, яка на швидкості вище 30 км/год, використовує дії водія щодо рульового управління, газу та гальма, а також динамічні дані про швидкість та обертання транспортного засобу, щоб визначити, чи вживаються надзвичайні дії, або якщо машина вийшла з-під контролю. Якщо так, серед інших заходів, система зменшує провисання ременів безпеки з використанням реверсивних натягувачів і, якщо боковий удар

або перевертання вважається високоймовірним, він закриває електричні вікна та люк. PRE-SAFE® Brake - це автономна система екстреного гальмування, яка додатково використовує радарні датчики, що допомогають виявити критичні ситуації.

При швидкості від 30 км / год до 200 км / год, відстань близько 200 метрів попереду автомобіля сканується на наявність радіолокаційних відбиваючих перешкод. Десь за дві з половиною секунди до цьогоу дару, водія попереджають про потенційну небезпеку. Якщо на цьому етапі водій застосовує гальмо, автомобіль автоматично подає гальмівну силу, необхідну для безпечного зупинення автомобіля (якщо це фізично можливо), незалежно від тиску, який чинить водій. Однак якщо критична ситуація продовжує розвиватися, а водій не реагує, машина застосовує часткове гальмування і за пів секунди до зіткнення та натягує ремені безпеки під час підготовки до удару. Нарешті, якщо PRE-SAFE® визначає, що зіткнення уникнути неможливо, автомобіль застосовує максимальне гальмування щоб максимально зменшити силу удару[2].

За оцінками, близько 19 відсотків усіх серйозних аварій в Німеччині пов'язані з наїздом ззаду.. Основні причини таких аварій - відволікання уваги або неуважність водія, а також неправильне сприйняття водієм при неправильній оцінці дорожньої ситуації. PRE - SAFE® працює на швидкостях від 30 км / год до 200 км / ч. Тому система більше орієнтована на аварії на високій швидкості. ніж типові системи для міського водіння, і очікувані переваги зміщені в бік серйозних і смертельних травм. Зниження на 35 відсотків числа людей, які отримали як мінімум важкі травми.

Однак це не означає, що треба повнісю покладатися виключно на цю технологію безпеки. На рисунку 1.2 зображено радарну систему автомобіля Mercedes.

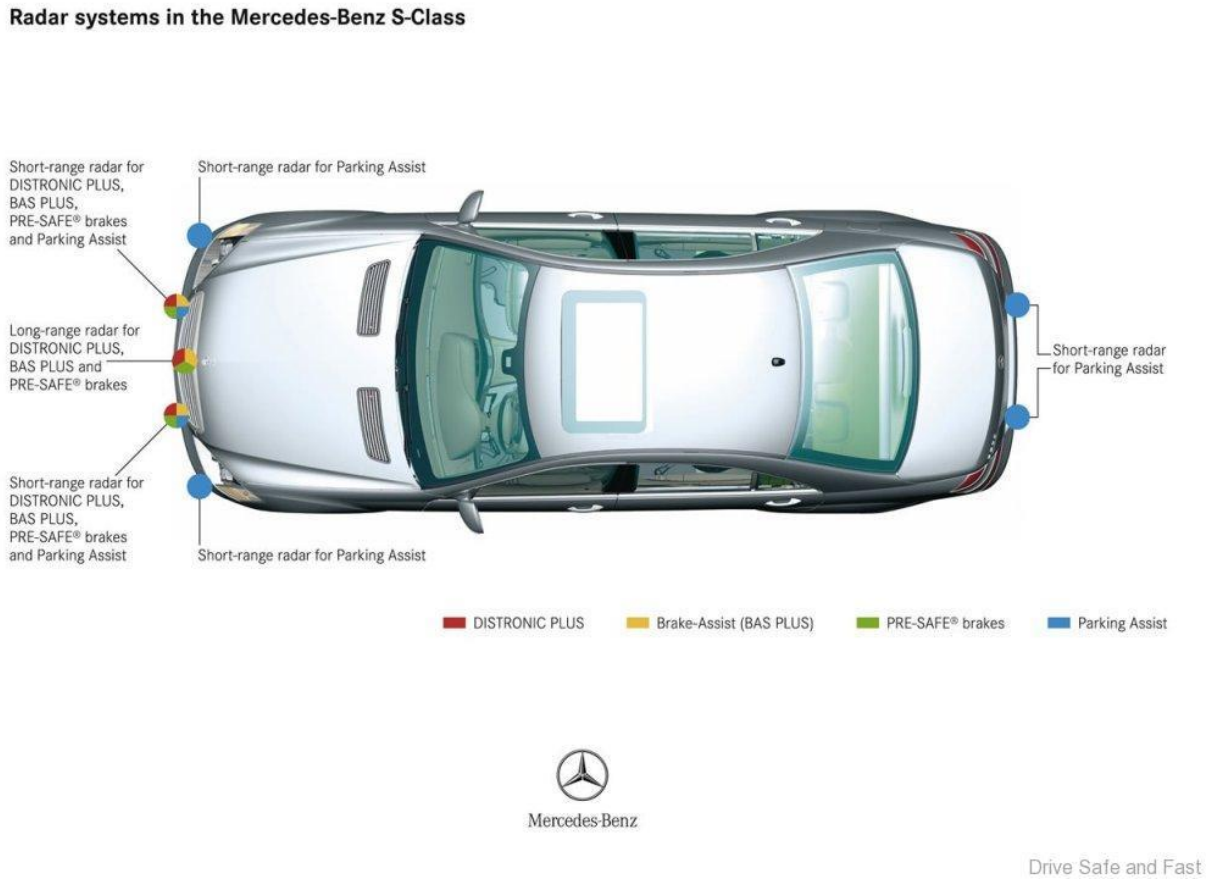

Рисунок 1.2 - Радарна система автомобілю[2]

1.3.2 Pre-collision Toyota

В системі Pre-collision використовується радар та камери для виявлення пішоходів та автомобілів.

У разі високої можливості зіткнення з певним об’єктом дуже висока, система видає попередження та активує гальмівну систему

Для попередження фронтального зіткнення необхідно щоб швидкість була від 10 до 180 км / год.

Для детекції пішохода, швидкість повинна бути в діапазоні до 80 км / год. На рисунку 1.3 показано розташування датчиків системи [3].

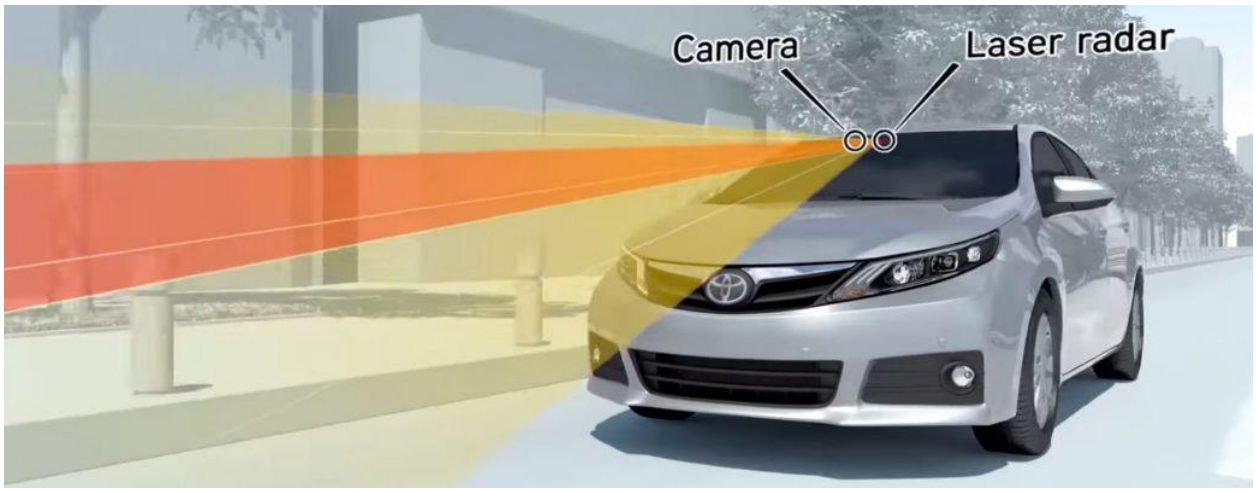

Рисунок 1.3 - Розташування датчиків системи Pre-Collision [3]

Доступно безліч версій системи запобігання зіткнень Toyota. З Toyota Safety Sense ™ P (TSS P) або Toyota Safety Sense ™ 2.0 (TSS 2.0) - найбільш поширеними версіями Toyota Pre-Collision також включає виявлення пішоходів.

Транспортні засоби з функцією виявлення пішоходів мають додаткову систему, встановлену на решітці радіатора, яка використовує радар міліметрового діапазону для перевірки вашого оточення на предмет більш дрібних об'єктів, які PCS може пропустити! Вивчаючи розмір, форму і рух цього меншого об'єкта, PD може ідентифікувати вразливих людей і при необхідності застосовувати гальма.

1.3.3 Pre-Sense Audi

Audi Pre-Sense - це технологічний пакет, який включає в себе систему допомоги водієві для підвищення безпеки на вулицях Ферхоуп. Всі версії нових моделей включають в себе систему безпеки Audi Pre-Sense в якості стандартного обладнання. На швидкості до 85 км / год (52,8 миль / год) система сканує дорогу на предмет наявності інших транспортних засобів і пішоходів за допомогою встановленої на лобовому склі фронтальної камери з дальністю дії до 100 метрів. Якщо існує загроза зіткнення, водій отримує серію попереджень, і при

необхідності автомобіль починає повністю гальмувати. На швидкості до 40 км / год він може повністю запобігати аваріям в межах системи. На більш високих швидкостях до 85 км / год попередження і застосування гальм можуть знизити швидкість удару.

Система допомоги при повороті використовує радарні датчики з переднім конусом для відстеження зустрічного транспорту при виконанні повороту направо або наліво. Система допомоги при повороті автоматично активується, коли автомобіль рухається з зупинки або повільно рухається зі швидкістю 10 кілометрів на годину. Функція допомоги при повороті активується, коли при спробі повернути зустрічний автомобіль виявляється на відстані зіткнення. Це автоматичне втручання задіє гальма, щоб запобігти зіткненню автомобіля з зустрічним автомобілем, утримуючи ваш автомобіль в межах своєї смуги руху. Індикатор на цифровий кабіні буде блимати, щоб повідомити вам про втручання системи допомоги при повороті.

1.3.4 Driving Assistant BMW

Системи BMW Active Driving Assistant допомагають водієві уникнути зіткнення. У пакет входять вдосконалені системи допомоги водієві (ADAS), які все частіше стають стандартним обладнанням як для основних, так і для люксових брендів.

Попередження про лобове зіткненні відстежує перешкоди попереду, а за допомогою Active Driving Assistant технологія також може визначати пішоходів на шляху транспортного засобу. Якщо водій не може загальмувати, спрацьовує автоматична система екстреного гальмування, щоб зупинити або сповільнити BMW перед зіткненням. Обов'язково варто розуміти, що з Active Driving Assistant автоматичне гальмування працює тільки на низьких міських

швидкостях. На швидкісних автомагістралях і автострадах водій повинен реагувати на можливі зіткнення.

Крім того, Active Driving Assistant включає систему попередження про сліпих зонах, яка відстежує ліву і праву сліпі зони автомобіля. Він попереджає водія візуальними і, при необхідності, звуковими сигналами, покликаними привернути увагу водія і переконати водія змінити план зміни смуги руху. Система попередження про перехресне русі ззаду використовує той же набір датчиків, щоб вказати водієві, коли транспорт наближається до BMW збоку, наприклад, при виїзді заднім ходом з паркувального місця.

Попередження про виїзд з смуги руху також є частиною BMW Active Driving Assistant. На більш високих швидкостях він контролює розмітку смуги руху і попереджає водія, якщо автомобіль ненавмисно з'їжджає з смуги руху. Зміна смуги руху при використанні покажчика повороту відключає систему попередження про виїзд з смуги руху.

На рисунку 1.4 показано приклад роботи системи.

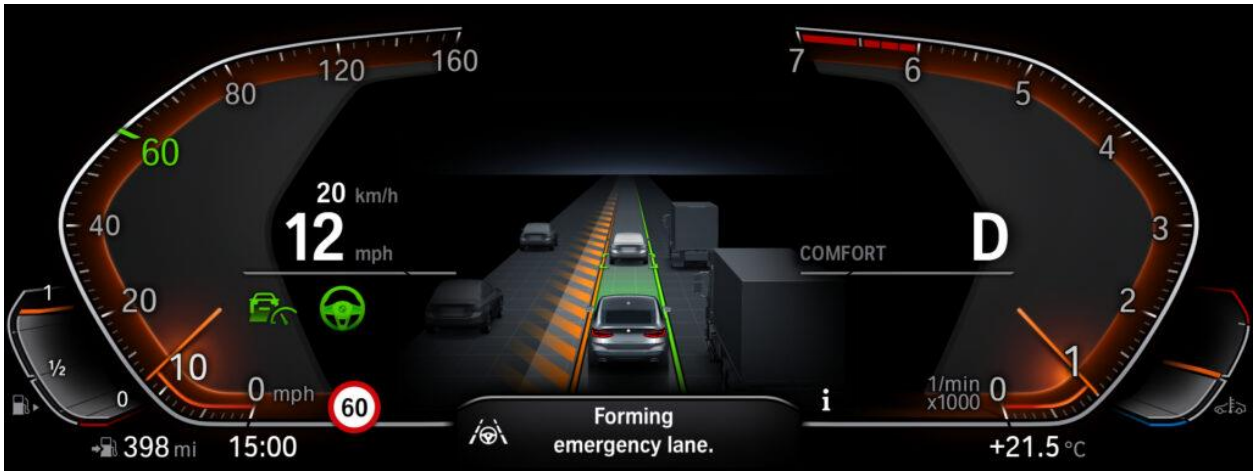

Рисунок 1.4 - Приклад роботи системи Driving Assistant Plus

1.3.5 Постановка задачі

Розроблювана система призначена для використання у галузі автомобілебудування та створення систем автономного управління транспортними засобами.

Головною метою системи є зменшення кількості дорожньо-транспортних пригод та збільшення безпеки дорожнього руху. А також зручності користування автомобілем.

Головною задачею що ставиться перед системою є забезпечення швидкого розпізнавання об’єктів, визначення їх положення та розмірів.

Для можливості використання у реальному житті необхідно забезпечити високий рівень надійності та якості роботи системи, а також забезпечити таку швидкодію щоб мати можливість проводити детекцію об’єктів у реальному часі.

Розроблювана система повинна базуватися на уніфікованих та загальноприйнятих стандартах обміну даними для реалізації можливості використання у існуючих транспортних засобах.

Висновок

В даному розділі були розглянуті наявні аналогічні системи від великих автовиробників. Також було досліджено різні типи систем оптичного розпізнавання, їх недоліки та переваги відносно одна одної. Зважаючи на інформацію, що було викладено у розділі, встановлено актуальність проблематики, що розглядається в магістерській дисертації та сформовано критерії яким вона повинна відповідати та які задачі виконувати.

2 ТЕОРЕТИЧНІ ОСНОВИ СИСТЕМИ РОЗПІЗНАВАННЯ ОБ'ЄКТІВ У ВІДЕОПОТОЦІ

2.1 Загальна характеристика

Останніми роками автономний транспорт здобуває все більшу популярність та одним з головних напрямків конкуренції серед автовиробників. Основними цілями при реалізації таких систем є забезпечення точності та швидкості розпізнавання.

Для забезпечення якості розпізнавання необхідно отримувати та обробляти дані з різних датчиків. Одним з завдань, що ставилися переді мною є огляд сучасних методів розпізнавання об'єктів та їх порівняння.

Хоча на перший погляд завдання детекції об'єктів на зображенні є схожим на задачу класифікації, але це не зовсім так. По-перше на початковому етапі невідомо скільки об'єктів шуканих класів містяться на зображенні та чи є вони там взагалі. По-друге нам невідоме розташування цих об'єктів. В результаті вирішення цих двох ключових проблем утворюється окремий клас задач машинного навчання, а саме розпізнавання об'єктів.

Завдяки розвитку глибинного навчання, та вдосконаленням методів, що в ньому використовуються вдалося значно покращити результати в порівнянні з більш традиційними методами машинного навчання. Підходом глибинного машинного навчання, що демонструє найкращі результати на даний момент та так би мовити state of the art в цій сфері є використання згорткових нейронних мереж, тому інші методи базуються саме на них.

На даний момент використовують два основних підходи в задачах розпізнавання об'єктів. Перший це розбиття вхідного зображення на так звані регіони інтересу, та їх подальша обробка та класифікація об'єктів, що потрапляють у ці регіони. Фактично цей метод містить в собі 2 окремих алгоритми які виконують різні задачі, через це швидкодія може бути на не досить високому рівні, особливо коли необхідне розпізнавання в реальному часі. Інший

підхід використовує одну мережу і для розбиття зображення, і на класифікацію об’єктів. Для цього використовують так звані anchor boxes, якщо в цей бокс потрапляє об’єкт, що потенційно відноситься до шуканого класу, то розраховується як необхідно змістити бокс та який клас об’єкту що в ньому знаходиться. Перші версії імплементацій цього підходу мали проблему з визначення дрібних об’єктів на зображенні, але в подальшому її вплив було значно зменшено.

2.2 Регіони інтересу

Виявлення регіонів інтересу є першим та найбільш важливим етапом для алгоритму виявлення об’єктів. На цьому етапі зазвичай виконують певну обробку зображень для полегшення виявлення регіонів інтересу. За допомогою застосування певних фільтрів виявляються краї об’єктів, патерни та інші. Безпосереднь для пошуку ROI використовують або селективний пошук, або метод переміщуваного вікна, або декорельваний канал характеристик.

Найпростішим, але й найповільнішим є метод послідовно переміщуваного вікні, його суть можна зрозуміти з назви: беруться рамки різних розмірів та послідовно переміщуються по зображенню та перевіряється чи потрапляють в них певні об’єкти. Очевидно, що цей процес є відносно довгим, оскільки кожен регіон перевіряється нейронною мережею на наявність об’єктів.

Селективний пошук полягає в тому, що генерується багато регіоні кожен з яких відноситься максимум одного класу. Потім з множини регіонів обирається 2 найбільш подібних, та об’єднуються в один. Даний процес повторюється до того поки не залишиться 1 регіон. Приклад проведення селективного пошуку наведено на рисунку 2.1

Рисунок 2.1 - Приклад виконання селективного пошуку

Метод декорельваного каналу характеристик поєднується з нейронно. Мережею в якій визначається рівень довіри Цей рівень вказує на те наскільки можлива наявність певного об'єкта в кадрі[6].

2.3 Отримання ознак

Загалом немає якихось обов’язкових алгоритмів для знаходження ознак та класифікації об’єктів і тому вони можуть бути написані самостійно. Досить часто для написання власних алгоритмів витягу ознак використовують гістограми орієнтованих градієнтів.

Опис зображення є проміжним, але вкрай важливим етапом рішення задач комп'ютерного зору. У процесі опису зображення відбувається перетворення візуальних даних, що містяться на зображенні, в прийнятну для алгоритму класифікації форму. Так як в візуальних даних зазвичай міститься багато зайвої інформації, то задача опису крім іншого полягає у виключенні якомога більшої кількості неінформативних ознак при збереженні даних про всі істотні для завдання розпізнавання характеристиках досліджуваного фрагмента

зображення. HOG дескриптори використовуються класифікаторами SVM, Random Forest, Boosting і багатьма іншими.

За великим рахунком алгоритм складається з двох частин: екстракція вектора ознак з даних (зображення) і навчання моделі.

Для аналізу зображення алгоритм використовує слайдинг вікно розміром 128 × 64 пікселів. Вікно переміщається по зображенню і формує вектор ознак для кожного положення вікна. Це відмінно працює для випадків, коли наш об'єкт ідеально вписується в вікно, що зустрічається далеко не завжди. Для вирішення цієї проблеми використовується так звана піраміда зображень (image pyramid). Слайдінг вікно сканує зображення кілька разів, після кожного проходу зображення стискується, при цьому розмір вікна залишається незмінним - 128 × 64 пікселів. Візуально це можна уявити як піраміду з зображень, як показано на рисунку 2.2.

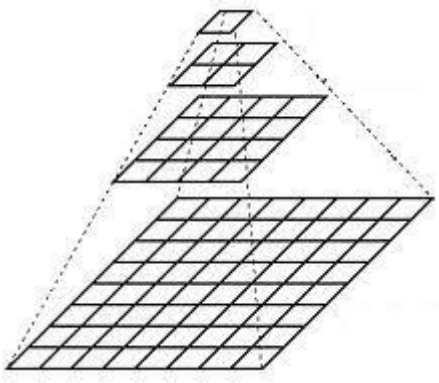

Рисунок 2.2 - Сканування за допомогою слайдинг вікна

Після цього розраховується градієнт зображення. Градієнт зображення дозволяє виділити найбільш важливі частини зображення і завуалювати менш важливі. Під важливістю розуміється зміна яскравості пікселів. Для обчислення градієнта використовуються фільтри, які застосовуються до зображення, автори алгоритму пробували різні фільтри, і виявилося, що найкращі результати виходять при використанні простих фільтрів:

$$(1\ 0\ 1),\ (-1\ 0\ 1)^T$$

Перший фільтр формує градієнт по горизонталі, другий по вертикалі. Для отримання модуля та напрямку вектору градієнта використовують стандартні розрахунки. Після цього розраховується безпосередньо гістограма градієнтів[7].

Зображення розбивається на осередки розміром 8 × 8, тобто 64 пікселя в осередку, з кожним пікселем асоційоване 2 значення, що описують градієнт: напрямок вектора градієнта і його модуль. З урахуванням розмірів слайдинг вікна у отримують 128 осередків на вікно, для кожного осередку формується гістограма градієнтів, яка представляє собою звичайний масив, що складається з 9 елементів (9 bins), іншими словами, відбувається розподіл модулів 64-х градієнтів на 9 елементів масиву. Виконується це в такий спосіб: 9 елементів масиву нумеруються з 0 до 160 з кроком 20, тобто 0, 20, 40 і т. Д. Величина модуля градієнта розподіляється між двома сусідніми осередками пропорційно вектору напрямку градієнта. Наприклад, якщо вектор напрямку градієнта 90 градусів і модуль дорівнює 120, тоді в осередку 80 та 100 додаються по 60.

На перший погляд незрозуміло, чому діапазон 0-180, адже напрямок вектора може бути в діапазоні 0-360 (або іншими словами 0 - / + 180) та навіщо потрібна гістограма градієнтів. Даний діапазон було обрано з практики. Після маси тестів, проведених авторами алгоритму, випливає, що якщо використовується діапазон 0-180, тобто знак нак вектору ігнорується, то величина помилки розпізнавання мінімальна. По другому питанню, перетворення градієнтів для пікселів в гістограмму градієнтів переслідує 2 цілі: зменшити розмірність вектора ознак (feature vector), що повинно добре позначитися на SVM класифікаторі, і основна мета - підвищити generalization нашого вектора ознак. Що під цим мається на увазі, якщо зображення піддається невеликий геометричній деформації, то це слабо позначається на гістограмі градієнтів, в тому числі і завдяки нормалізації гістограми градієнтів, описаної нижче. Гістограма градієнтів, отримана на попередньому кроці, змінюється в залежності від освітленості об'єкта. Збільшення освітленості в 2 рази призведе до пропорційного збільшення величини градієнтів гістограми, така варіативність від освітлення може серйозно вплинути на якість навчання моделі. В ідеалі

освітленість не повинна впливати на гістограму і для цієї мети використовується нормалізація, а саме перетворення гістограми градієнтів в одиничний вектор (вектор з модулем рівним одиниці). У нашому випадку гістограма являє собою вектор розмірністю 9 і щоб перетворити його в одиничний вектор потрібно виконати наступні обчислення:

$$mod = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2} \quad (1)$$

$$x'_n = x_n / mod \quad (2)$$

Таким чином результуючий вектор має вигляд ($x`_0$, $x`_1$, …, $x`_n$).

Цей підхід буде працювати, але автори алгоритму запропонували ідею, яка показала кращі результати. ЇЇ суть полягає у тому, що нормалізується не окрема гістограма, а блок розміром 2x2, що складається з 4 комірок. Блок переміщується по зображенню з перекриттям, для кожного положення блоку виконується конкатенація гістограми градієнтів для комірок у блоці та виконується нормалізація результуючої гістограми. В результаті для кожної комірки формується 4 гістограми.

Методи глибинного машинного навчання дозволяють мережі визначати функції, які неможливо реалізувати власноруч. Це може забезпечити більш високий рівень абстракції. Залежно від застосування можуть використовуватися різні методи. Для кожної вхідної області створюється вектор дійсних або довічних значень. Вихідний вектор являє собою видимі характеристики пропонованих регіонів.

2.4 Класифікація

Отриманий вектор ознак подається в класифікатор для визначення наявності об’єкту в регіоні уваги. Класичним прикладами таких класифікаторів є SVM або багатошаровий перцептрон.

Нейронні мережі із прямим зв’язком складаються з серії обчислювальних вузлів (нейронів), що пов'язані між собою для обробки інформації. Така мережа має назву багатошаровий перцептрон (MLP). Багато вузлів утворюють шари, що з'єднані між собою сила зв’язку між шарами оцінюється вагами. Нейрон функціонує як класифікатор логістичної регресії. В нейронах використовуються нелінійні операції для перетворення вхідних даних та створення границі рішення, в якій дані можуть бути лінійно відокремлювані. Зображення одиничного перцептрона показано на рисунку 2.3. Кілька шарів цих перцептронів створюють MLP або нейронну мережу (рисунок 2.4). Нейронна мережа на рисунку 2.4 є повністю зв’язаною мережею. Це означає, що кожен нейрон наступного шару отримує вхід від кожного нейрона з попереднього шару.

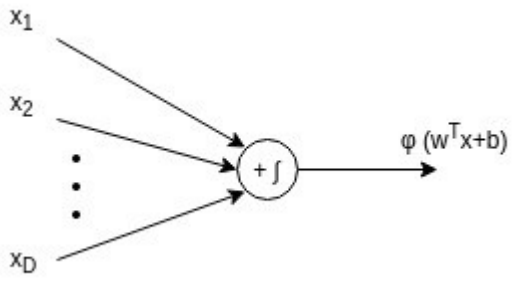

Рисунок 2.3 - Зображення перцептрону

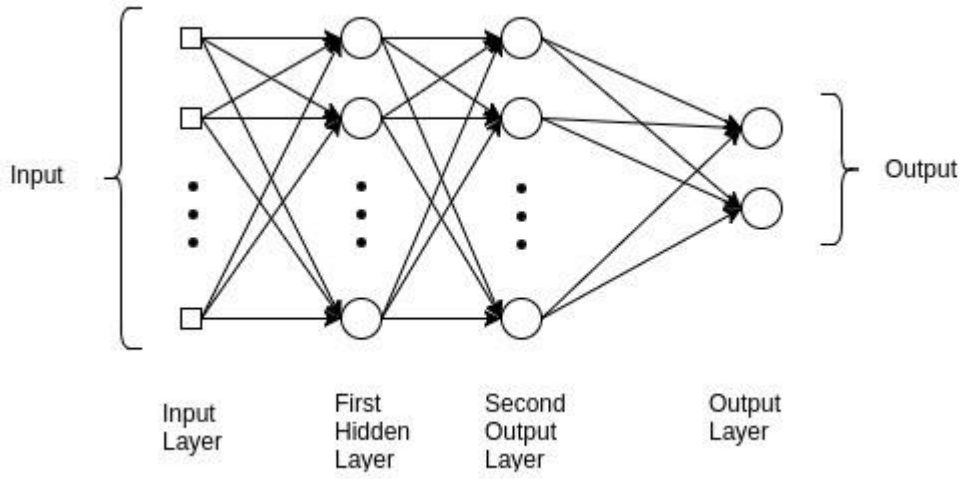

Рисунок 2.4 – Повнозв’язна нейронна мережа

Останнім часом, класичний підхід виявлення ознак вручну втрачає популярність в зв’язку із розвитком згорткових нейронних мереж, що дають значно кращі результати під час навчання. Згорткові нейронні мережі відносяться до так званого глибинного навчання.

Глибинне навчання - це метод машинного навчання, який вчить комп'ютери робити те, що природно для людей: вчитися на власному прикладі. Глибинне навчання - це ключова технологія, що лежить в основі безпілотних автомобілів, що дозволяє їм розпізнавати знак зупинки або відрізняти пішохода від ліхтарного стовпа. Це ключ до голосового управління в споживчих пристроях, таких як телефони, планшети, телевізори. Останнім часом глибинному навчанню приділяється багато уваги і не дарма. Це досягнення результатів, які раніше були неможливі.

При глибинному навчанні комп'ютерна модель вчиться виконувати завдання класифікації безпосередньо із зображень, тексту або звуку. Моделі глибинного навчання можуть досягати найвищої точності, що іноді перевищує продуктивність людського рівня. Моделі навчаються з використанням великого набору розмічених даних і архітектур нейронних мереж, які вивчають функції безпосередньо з даних без необхідності вилучення функцій вручну.

Термін «глибокий» зазвичай відноситься до кількості прихованих шарів в нейронної мережі. Традиційні нейронні мережі містять всього 2-3 приховані шари, в той час як глибинні мережі можуть мати до 150.

Класичним прикладом згорткової нейронної мережі є EfficientNet від Google. З тих пір, як AlexNet виграла змагання ImageNet в 2012 році, CNN (Convolutional Neural Networks) стали де-факто алгоритмами для широкого кола завдань глибокого навчання, особливо для комп'ютерного зору. З 2012 року по сьогоднішній день дослідники експериментують і намагаються придумати все більш досконалі архітектури для підвищення точності моделей для різних завдань. EfficientNet, в якій основна увага приділяється не тільки підвищенню точності, але і ефективності моделі. EfficientNet встановила нові рекорди як за точністю, так і по ефективності обчислень.

Загалом підвищення точності склало до 6%, при цьому ефективність близько 5-10 разів вище, ніж у більшості сучасних CNN. Порівняння EfficentNet з існуючими архітектурами наведено на рисунку 2.5

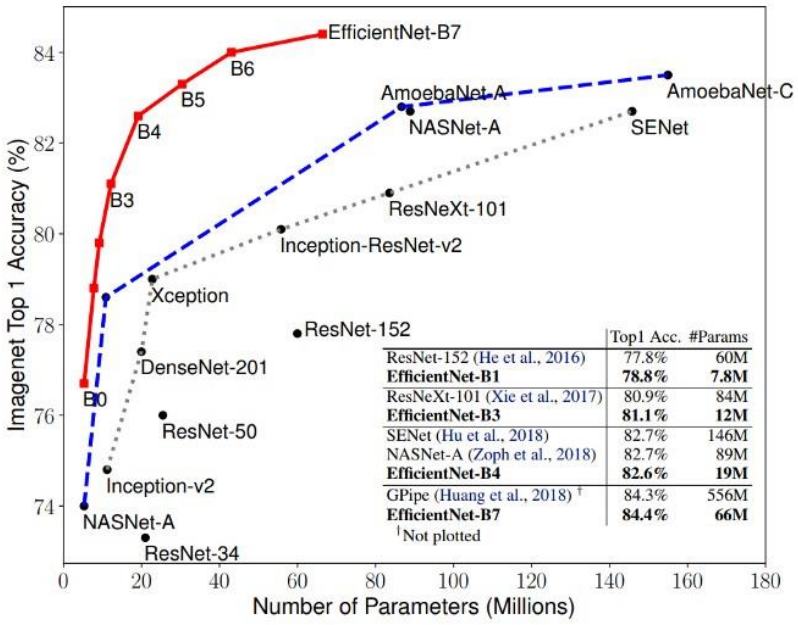

Рисунок 2.5 - Порівняння EfficentNet з існуючими архітектурами

2.5 Згорткові нейронні мережі

Згорткова нейронна мережа (ConvNet / CNN) - це алгоритм глибокого навчання, який може приймати вхідне зображення, присвоювати важливість (ваги і зміщення) різним об'єктам на зображенні і мати можливість відрізняти один об’єкт від іншого. Попередня обробка, необхідна в згорткових мережах, набагато нижче в порівнянні з іншими алгоритмами класифікації. У той час як в примітивних методах фільтри створюються вручнуі, ConvNets можуть вивчати ці фільтри самостійно.

Архітектура ConvNet аналогічна структурі сполуки нейронів в людському мозку і була натхненна організацією візуальної кори. Окремі нейрони реагують на стимули тільки в обмеженій області поля зору, відомої як поле сприйняття. Множина таких полів перекривається, щоб покрити всю візуальну область.

Для отримання характеристик об’єктів використовуються згорткові операції, що в загальному вигляді описуються як:

$$m(t) = (x * k)(t) = \sum_{a=-\infty}^{a=+\infty} x(a)\, k(a + t) \qquad (3)$$

Оскільки за межами поля сприйняття значення детектора ознак вважається нульовим, нескінченну суму можна замінити скінченною (сума значень у даній області). Після використання властивості комутативності математичної згортки, отримуємо спрощнну функцію, що споріднена з взаємокориляційною функцією для зображень і яка придатна для створення детекторів згорткового шару CNN:

$$M(i,j) = (K * X)(i,j) = \sum_m \sum_n K(m,n) X(i - m, j - n) \qquad (4)$$

де $M(i,j)$ – елемент карти ознак з координатами i та j;

X – вхідне зображення;

K – детектор ознак;

m, n – розмірності детектора ознак.

Мета операції згортки полягає у вилученні високоякісних ознак, таких як краї, із вхідного зображення. Згортковим мережам не потрібно обмежуватися лише одним згортковим шаром. Зазвичай, перший згортковий шар відповідає за екстракцію ознак низького рівня, таких як краї, колір, орієнтація градієнта та інших. За допомогою доданих шарів архітектура також адаптується до ознак високого рівня, надаючи нам мережу, яка має цілісне розуміння зображень у наборі даних, подібно до того, як ми б це зробили.

На згортковому шарів ядро послідовно застосовується до кожного з каналів зображення (зазвичай по 1 каналу для червоного, зеленого, синього кольорів). В ході цього “сканування” виконується перемноження вхдних даних із значеннями в фільтрі. Потім результат сумується та записується в результуючу матрицю, а фільтр зміщується на певний крок. Таким чином якщо певна ознака потрапляє у зону ядра, вона буде визначена. Розмір ядра впливає на кількість ознак, що об’єднуються для отримання нової ознаки.

Існує два типи результатів операції на згортковому шарі - в одному випадку згорнутий елемент на виході зменшується порівняно з вхідним, і другий, у якого розмірність або збільшується, або залишається незмінною. Це робиться шляхом застосування valid padding у випадку першого, або же same padding у випадку другого.

Коли ми збільшуємо зображення 5x5x1 до зображення 6x6x1, а потім застосовуємо над ним ядро 3x3x1, ми бачимо, що згорнута матриця виявляється розміром 5x5x1. Звідси і назва - same padding.

З іншого боку, якщо ми виконуємо ту саму операцію без доповнення (padding), ми отримуємо матрицю, яка має розміри самого ядра (3x3x1), - valid padding.

В згорткових мережах також використовуються і так звані pooling шари. Цей шар відповідає за зменшення просторового розміру матриці ознак. Такий підхід має зменшити обчислювальну потужність, необхідну для обробки даних, завдяки зменшенню розмірності. Крім того, це корисно для вилучення домінантних ознак, які є інваріантними щодо обертання та позиції, таким чином підтримуючи процес ефективного навчання моделі.

Існує два типи пулінгу: max pooling та average pooling. Перший повертає максимальне значення частини зображення, охопленої ядром. З іншого боку, Середнє об’єднання повертає середнє значення всіх значень із частини зображення, охопленої ядром.

Max pooling також виконує функцію зменшення рівня шуму. Він взагалі відкидає шумові активації, а також виконує зменшення рівня шуму разом із зменшенням розмірності. З іншого боку, середнє об’єднання просто використовує зменшення розмірності як механізм зменшення шуму. Отже, ми можемо сказати, що max pooling працює набагато краще, ніж average pooling.

Згортковий шар та pooling шар разом утворюють і-й шар згорткової нейронної мережі. Залежно від складності зображень, кількість таких шарів може бути збільшена для подальшого захоплення деталей низького рівня, але ціною більшої обчислювальної потужності.

Пройшовши вищевказаний процес, ми успішно дозволили моделі виявити ознаки, що в подальшому передаються до класифікатора, в якості якого зазвичай використовують багатошаровий перцептрон.

Алгоритми глибинного навчання, що використовують для розпізнавання об'єктів поділяються на дві категорії: одноетапні детектори, що використовують нерегіональний підхід до пропозиції регіону, та двоетапні, що використовують двоетапний підхід. Метою одноетапного детектору є усунення необхідності у вилученні регіонів інтересу шляхом виконання цих дій в одній мережі. Суть його полягає в тому, що спочатку знаходяться регіони інтересу, об’єкти в яких потім класифікуються глибинною мережею. Цей підхід є більш простим у навчанні та є більш ресурсоефективним.

Детектори, що базуються на регіональному підході складаються з 2 частин, а глибинною мережею в них є лише класифікатор (зображено на рисунку 2.6). Вони є більш важкими для навчання через необхідність виявлення ознак, що передаються класифікатору. Схеми роботи мереж заснованих на регіональному та нерегіональному підході зображені на рисунку 2.7.

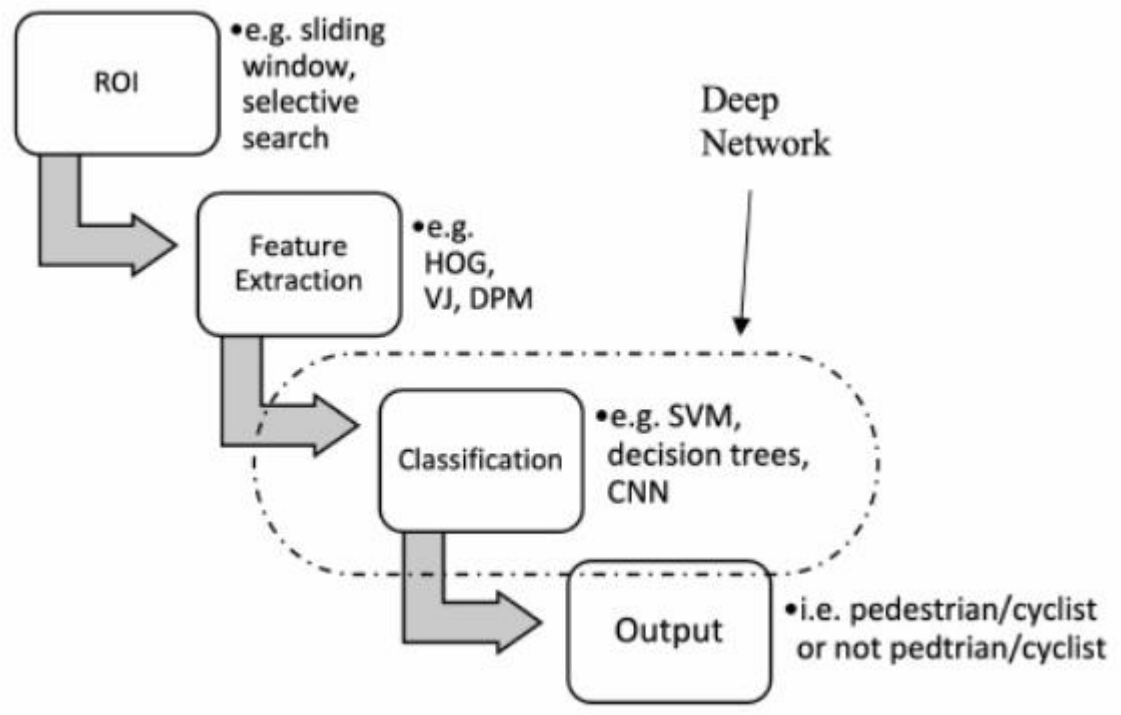

Рисунок 2.6 - Метод, що використовує регіональний підхід

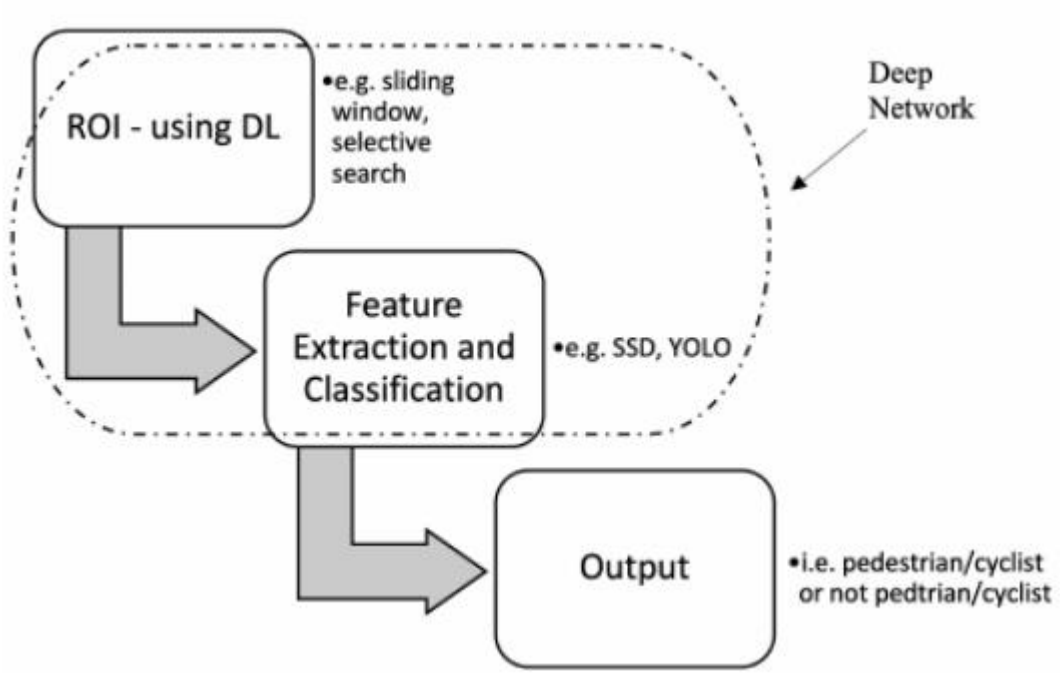

Рисунок 2.7 - Метод, що використовує не регіональний підхід

2.6 Двоетапні детектори

Згорткові нейронні мережі на основі пропозицій регіону продемонстрували позитивні результати в задачах виявлення об’єктів. Нейронні мережі, що відносяться до цього типу використовують різні підходи для виявлення регіонів інтересу, приклади цих підходів було наведено вище. Якість виявлення цих регіонів суттєво впливає на якість всієї мережі в цілому, оскільки саме результати з пошуку RoI передаються до класифікаторів. Для пришвидшення роботи таких детекторів було вирішено проводити виявлення ознак саме з цих регіонів, на даний момент саме такий підхід доволі успішно і застосовується для двоетапних детекторів і саме такий підхід дозволив збільшити середню точність отриману такими детекторами на 9%. Поява Fast R-CNN Faster та пізніше R-CNN дозволили також і підвищити швидкість розпізнавання. В R-CNN використовується селективний пошук для побудови 2000 пропозицій регіону, які потім подаються до згорткової нейронної мережі

для виявлення ознак, на основі яких потім проводиться класифікація. Очевидно що класифікація на основі такої великої кількості регіонів потребує значних витрат часу.

Хоча R-CNN мережі й використовують згортку для зменшення обчислювальних витрат, але дані детектори всеодно не можуть використовуватися для розпізнавання в реальному часі, щоб вирішити цю проблему було розроблено region proposal network, яка ділиться ознаками з мережею для виявлення об’єктів. RPN являє собою згорткову нейронну мережу, що одночасно передбачає границі об’єктів та проводить їх оцінку, цей підхід дозволив збільшити швидкість та точність виявлення об’єктів.

2.6.1 R-CNN

Робота мережі R-CNN складається з кількох єтапів:

1) преднавчання згорткової нейронної мережі для задачі класифікації зображень. Наприклад тренування VGG або ResNet на ImageNet датасеті;

2) відбувається пропозиція регіонів які шукаються методом селект вного пошуку. Дані регіони різних розмірів та потенційно можуть містити об’єкти необхідних класів;

3) регіони інтересу приводяться до необхідного для згорткової мережі розміру;

4) продовжити точну настройку CNN на відформатованих регіонах для K+1 класів. Додатковий клас відноситься до фону (об’єкти, що не є предметом детекції). На етапі тонкої настройки ми повинні використовувати набагато меншу learning rate, а mini-batch виконує надлишкову вибірку для позитивних випадків, тому що більшість пропонованих областей є лише фоном;

5) враховуючи кожну область зображення, одне проходження через CNN генерує вектор ознак. Потім цей вектор ознак споживається бінарним SVM, який навчається для кожного класу самостійно. Позитивні вибірки - це запропоновані області з порогом перекриття IoU ≥ 0,3, а негативні вибірки не мають значення є нерелевантними;

6) для зменшення помилок локалізації регресійна модель тренується корегувати вікно з урахуванням ознак згорткової мережі

На рисунку 2.8 зображено процес роботи R-CNN мережі

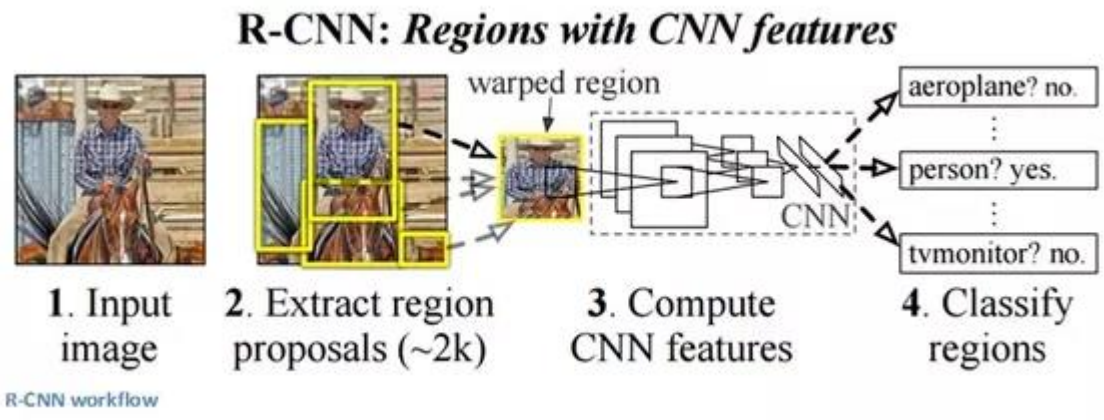

Рисунок 2.8 - Робота R-CNN мережі

Варто розглянути процес регресії обмежуючих рамок. З огляду на прогнозовану координату прямокутника (координата центру, ширина, висота) і відповідні йому координати істиної рамки, регресор налаштований на навчання масштабно-інваріантного перетворення між двома центрами і перетворення в логарифмічному масштабі між шириною і висотою. Всі функції перетворення приймають *p* в якості вхідних даних. Стандартна регресійна модель вирішує задачу мінімізації SSE. Параметр регуляризації в ході навчання має вирішальне значення, і в статті RCNN[8] було вибрано кращий параметр шляхом перехресної перевірки. Також слід зазначити, що не всі передбачені обмежуючі рамки мають відповідні істинні рамки. Наприклад, якщо немає перекриття, немає сенсу запускати регресію. Для навчання моделі регресії bbox зберігається тільки передбачений блок з значенням перетину мінімум 0,6.

Також в ході навчання R-CNN використовуються підхід, що має назву Hard Negative Mining. Суть його полягає в тому, що ми розглядаємо обмежуючі прямокутники без об'єктів як негативні приклади. Не всі негативні приклади однаково складно ідентифікувати. Наприклад, якщо він містить чистий порожній фон, це, швидше за все, «easy negative»; але якщо рамка містить зашумлену текстуру або частину об'єкту, його може бути важко розпізнати, і це «hard negative».

Hard negative приклади легко помилково класифікувати. Ми можемо явно знайти ці хибнопозитивні зразки під час циклів навчання і включити їх в тренувальні вибірки, щоб поліпшити класифікатор.

Переглядаючи етапи навчання R-CNN, легко виявити, що навчання моделі R-CNN є дорогим і повільним, оскільки наступні етапи включають в себе багато роботи:

– проведення вибіркового пошуку для пропозиції 2000 регіонів-кандидатів для кожного зображення;

– створення вектора характеристик CNN для кожної області зображення (N зображень * 2000).

– весь процес включає в себе три моделі окремо без великої кількості загальних обчислень: згорткову нейронну мережу для класифікації зображень і виділення ознак; класифікатор SVM для ідентифікації цільових об'єктів; і регресійна модель для корекції обмежувальних рамок області.

2.6.2 Fast R-CNN

Хоча в момент створення R-CNN її результати були досить високими, але як було зазначено вище, ця архітектура мала значні проблеми із швидкодією. Щоб зробити R-CNN швидше, Гіршік модифікував процедуру навчання,

об'єднавши три незалежні моделі в одну і збільшивши загальні результати обчислень, нова модель отримала назву Fast R-CNN. Замість того, щоб витягувати вектори ознак CNN незалежно для кожної пропозиції регіону, ця модель агрегує їх в один прямий прохід CNN по всьому зображенню, і пропозиції регіонів спільно використовують цю матрицю ознак. Потім ця ж матриця ознак використовується для навчання класифікатора об'єктів і регресорів обмежуючої рамки. Таким чином, спільне використання обчислень прискорює R-CNN.

Бінарні SVM в новій архітектурі не використовувалися, замість цього отримані ознаки передавалися на повнозв’язний шар, а потім на два паралельних шари: softmax з K + 1 виходами (додатковий клас для фону) і bounding box regressor, цей процес демонструється на рисунку 2.9.

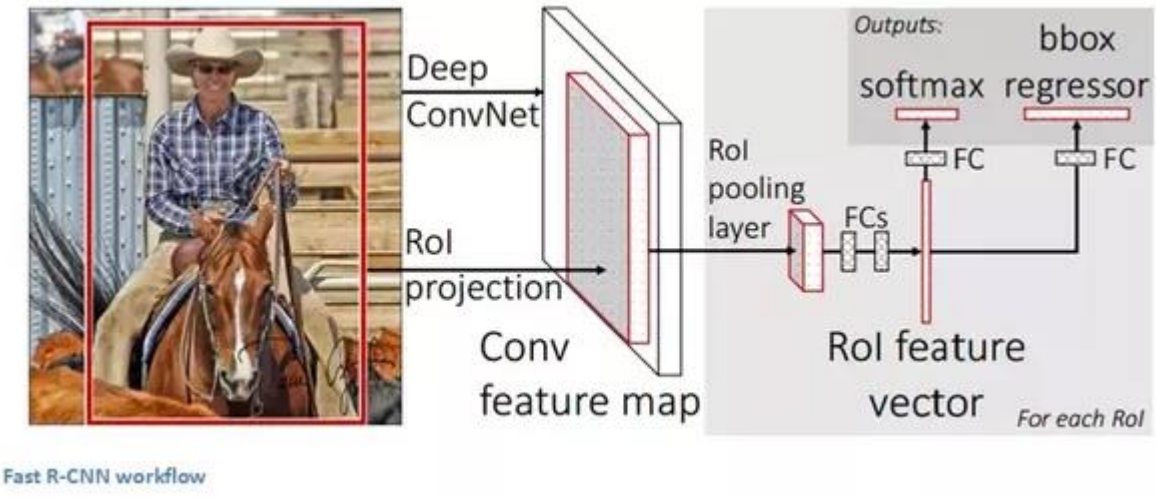

Рисунок 2.9 - Робота Fast R-CNN мережі

Для спільного навчання softmax-класифікатора і bbox regressor-а використовувалася об'єднана loss-функція:

$$L(p, u, t\,u\,, v) = Lcls(p, u) + \lambda[u \geq 1]Lloc(t\,u\,, v) \quad (5)$$

де u – клас об'єкта, реально зображеного в регіоні-кандидата;

$v = (v_x, v_y, v_w, v_h)$;

$Lcls(p, u) = -\log pu$ – логарифм loss для класу u;

$v = (v_x, v_y, v_w, v_h)$ – реальні зміни рамки регіону для більш точного охоплення об'єкта;

$t\,u = (tx\,u, ty\,u, tw\,u, th\,u)$ - передбачені зміни рамки регіону;

$Lloc$ – loss-функція між передбаченими і реальними змінами рамки;

$[u \geq 1]$ – індикаторна функція, рівна 1, коли $u \geq 1$, і 0, коли навпаки;

λ – коефіцієнт, призначений для балансування вкладу обох loss-функцій в загальний результат. У всіх експериментах він однаковий та дорівнює 1.

В загальному вигляді процес навчання складається з наступних етапів:

1) так само проводиться преднавчання згорткової мережі;
2) методом селективного пошуку генеруються регіони інтересу;
3) в згортковій мережі замість останнього макс пулінг шару використовується RoI пулінг шар, який повертає вектор ознак для регіонів інтересу. Спільне використання обчислень CNN покращує швидкість навчання, оскільки багато регіональних пропозицій для однакових зображень сильно перекриваються. Також додається ще один клас для фону;
4) результати попередніх операцій передаються на 2 вихідні шари: softmax та bbox regressor.

Fast R-CNN набагато швидший як під час навчання, так і під час тестування. Однак поліпшення выдносно базовоъ архытектури не є суттєвим, оскільки регіональні пропозиції генеруються окремо іншою моделлю, що дуже дорого.

2.6.3 Faster R-CNN

Інтуїтивним рішенням для прискорення мережы є інтеграція алгоритму регіональної пропозиції в модель CNN. Faster R-CNN робить саме це: будує єдину уніфіковану модель, що складається з RPN (мережі пропозиції регіонів) та

Fast R-CNN із загальними згорнутими функціональними шарами. Нова архітектура має назву Faster R-CNN та показана на рисунку 2.10

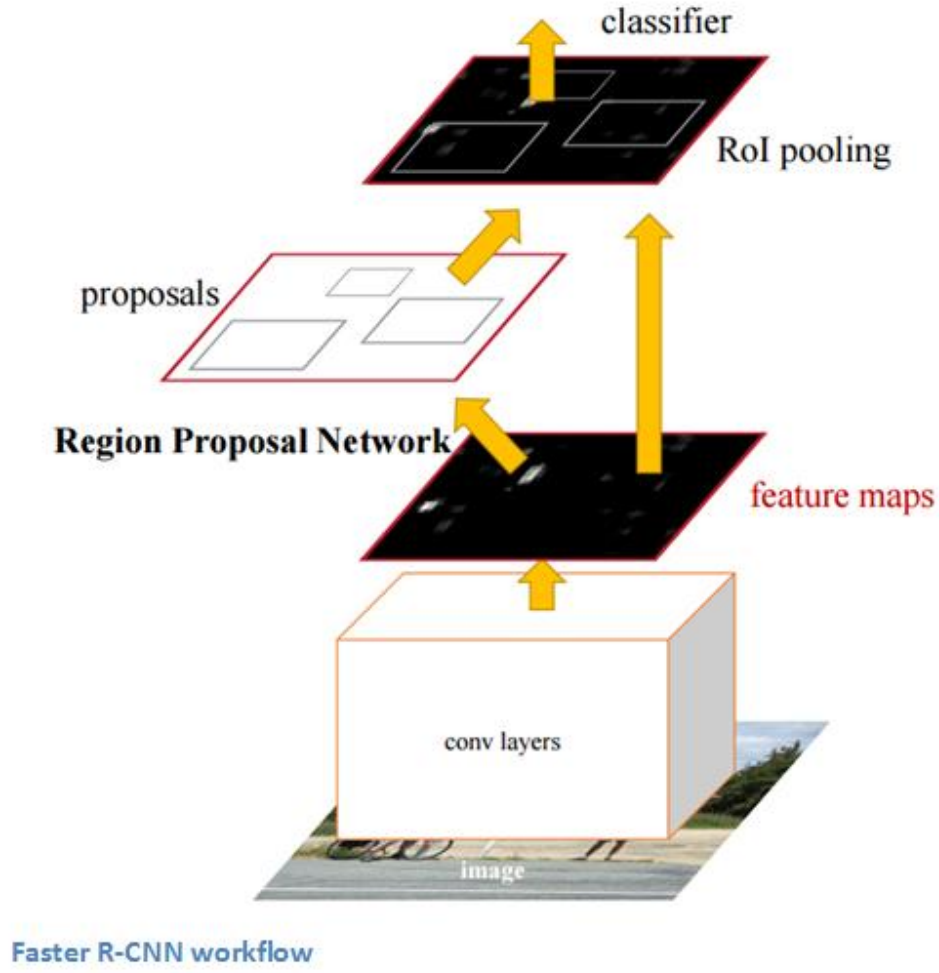

Рисунок 2.10 - Faster R-CNN мережа

В рамках RPN за отриманими CNN ознаками використовують малі нейромережі з невеликим (3x3) вікном. Отримані за допомогою цих мереж значення передаються в два паралельних повнозв'язних шари: box-regression layer (reg) і boxclassification layer (cls). Виходи цих шарів базуються на так званих anchor-ах: k рамках для кожного положення ковзного вікна, кожна з рамок має різні розміри і пропорції сторін. Reg-шар для кожного такого anchor-а видає 4 координати (координати центру та розміри вікна), вони застосовуються для корегування положення охоплюючої рамки; cls-шар видає два числа - ймовірності того, що в рамці знаходиться хоча б якийсь об'єкт або що не міститься жодного. Алгоритм продемонстровано на рисунку 2.11.

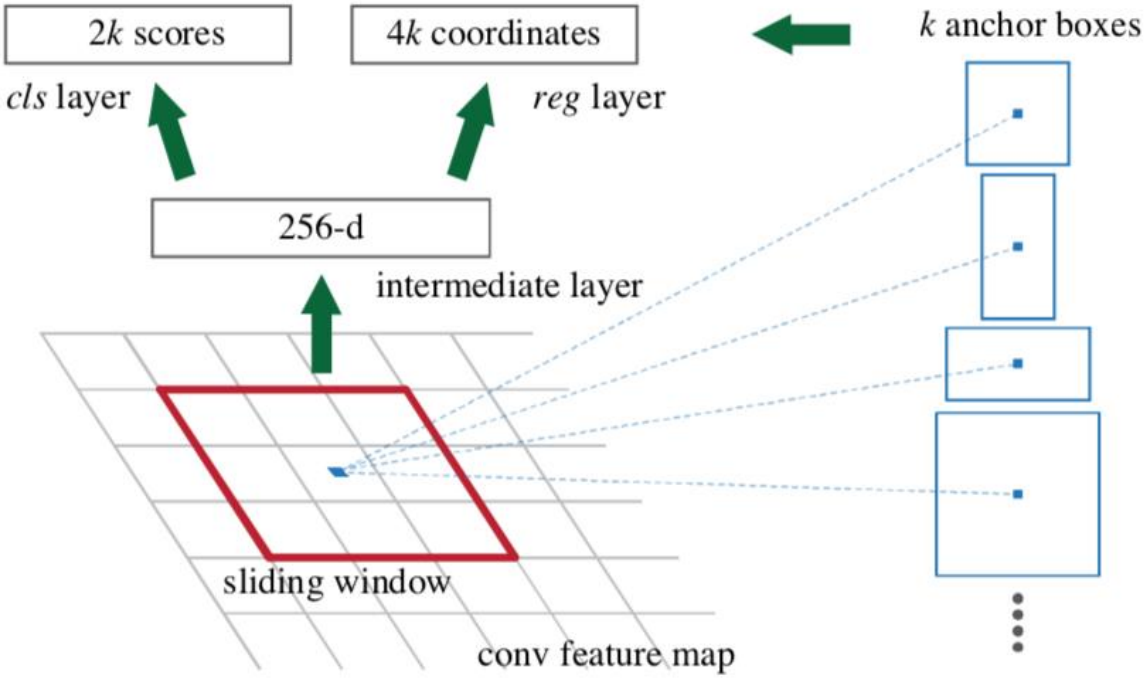

Figure 3: **Left**: Region Proposal Network (RPN).

Рисунок 2.11 - Алгоритм побудови карти ознак

Процес навчання reg і cls шарів об'єднаний; loss-функцію вони мають загальну, вона являє собою суму loss-функцій кожного з цих шарів, з застосуванням балансуючого коефіцієнту[12].

Обидва шари RPN видають тільки пропозиції для регіонів-кандидатів. Ті з них, регіони в яких з високою ймовірністю знаходиться об’єки, передаються далі в модуль детектування об'єктів і корекції охоплюючої рамки, який реалізований так само як і в Fast R-CNN.

Для того, щоб розділяти ознаки, одержувані в CNN, між RPN і модулем детектування, процес навчання всієї мережі побудований ітераційно, з використанням декількох кроків:

1) тоочне налаштування RPN (регіональної мережі пропозицій), яке ініціалізується преднавченим класифікатором зображень. Позитивні зразки мають IoU (перетин над об'єднанням) ≥ 0.7;

2) вікно n x n ковзає по карті ознак всього зображення. По центру кожного вікна передбачаються множина регіонів різних розмірів;

3) навчається модель Fast R-CNN, використовуючи пропозиції, сформовані RPN;

4) потім використовується мережа Fast R-CNN для навчання RPN. Зберігаються спільні згорткові шари, а точно налаштуються лише шари, характерні для RPN. На цьому етапі RPN і мережа виявлення мають спільні згорткові шари;

5) налаштовуються шари Fast R-CNN.

Запропонована схема не є єдиною, і навіть в поточному вигляді вона може бути продовжена за рахунок повтореня кроків 4 та 5.

2.7 Одноетапні детектори

Всі наведені вище нейронні мережі мають кілька суттєвих недоліків. По-перше вони дивляться на зображення не загалом, а лише на окремі регіони. По-друге вони відносно повільні. На відміну від них, YOLO не має цих недоліків. В перших блоках, ця архітектура не надто відрізняється від інших детекторів[13]. В ній так само за допомогою згорткової нейромережі отримується карти ознак, після аналізу яких отримуються bounding boxes (їх розміри позиції та класи).

Розглянемо в загальному вигляді процес навчання YOLO мережі.

а) зазвичай зображення зменшують для можливості формування батчів та пришвидшення навчання;

б) зображення розбиваються на клітинки, в YOLOv3 та v4 прийняте розбиття на клітинки розміром 13x13. Такі клітини, що називаються grid cells, лежать в основі ідеї YOLO. Кожна клітина є «якорем», до якого прикріплюються bounding boxes. Тобто навколо клітини малюються кілька прямокутників для визначення об'єкта (оскільки незрозуміло, якої форми прямокутник буде найбільш підходящим, їх малюють відразу кілька і різних форм), і їх позиції, ширина і висота обчислюються щодо

центру цієї клітини. На рисунку 2.12 наведено приклад розбиття зображення;

в) після цього наше зображення проходить через нейронну мережу і відбувається безпосередньо процес навчання.

Варто зазначити, крім картинки в тренувальному датасеті у нас повинні бути визначені позиції і розміри справжніх bounding boxes для об'єктів, які є на ній. Це називається «анотація» і робиться це зазвичай вручну.

Тепер варто розглянути процес навчання більш детально.

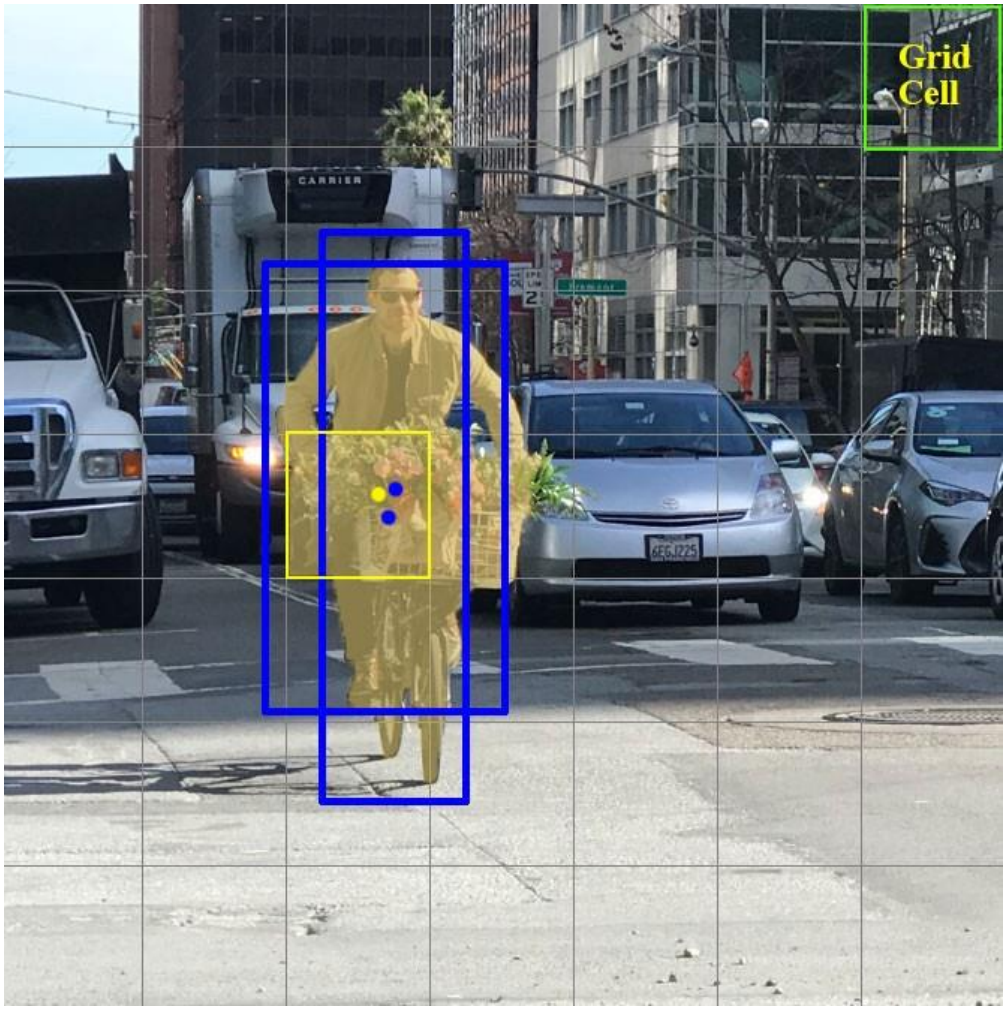

Рисунок 2.12 - Приклад розбиття зображення та встановлення anchor boxes

Розмір та положення bounding boxes відносно клітинки визначається за допомогою так званих anchor boxes. Вони задаються в самому початку або самим користувачем, або їх розміри визначаються виходячи з розмірів bounding boxes,

які є в датасеті, на якому буде тренуватися YOLO (використовується K-means clustering і IoU для визначення найбільш підходящих розмірів.)[14] Зазвичай задають близько 3 різних anchor boxes, які будуть намальовані навколо (або всередині) однієї клітини:

Під час навчання нам необхідно визначити дві речі:

а) який з anchor boxes, з 3 намальованих навколо клітини, нам підходить найбільше і як його можна трохи підправити для того, щоб він добре вписував в себе об'єкт;

б) який об'єкт знаходиться всередині цього anchor box і чи є він взагалі;

в) Таким чином на виході для кожної клітинки ми хочемо отримати наступні дані (зображено на рисунку 2.13).

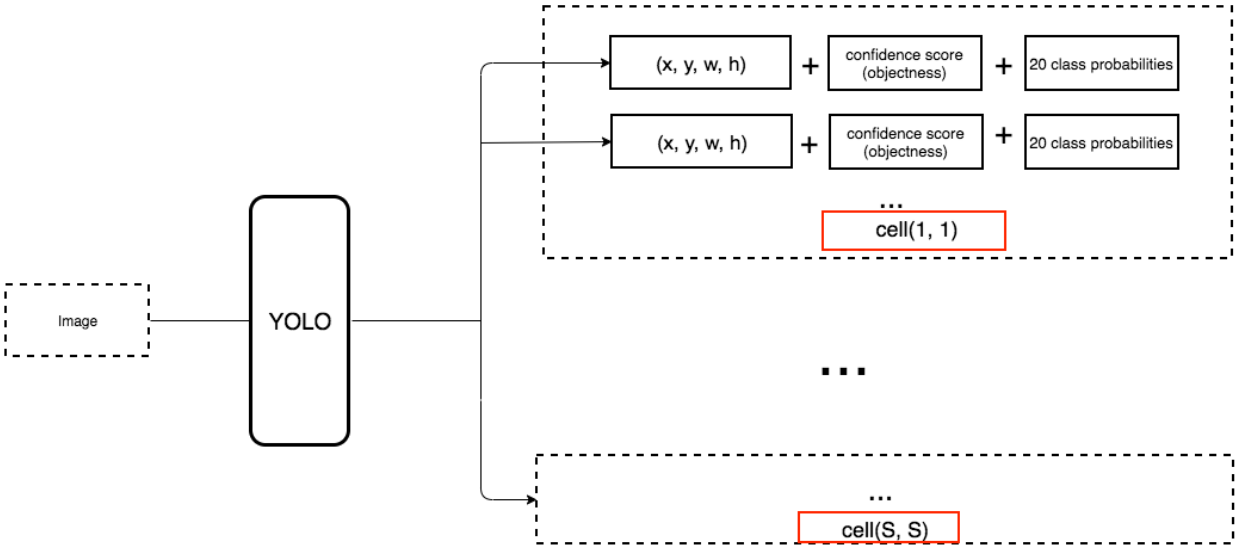

Рисунок 2.13 - Вихідні дані grid cell

Вихід умовно є сумою трьох складових:

а) x, y. w. h — позиція кожного anchor box відносно клітинки;

б) confidence score (objectness) — “якість” anchor box, що визначається за допомогою метрики IoU під час навчання;

в) class probability — ймовірнісний розподіл для кожного класу;

Як працює метрика IoU зображено на рисунку 2.14

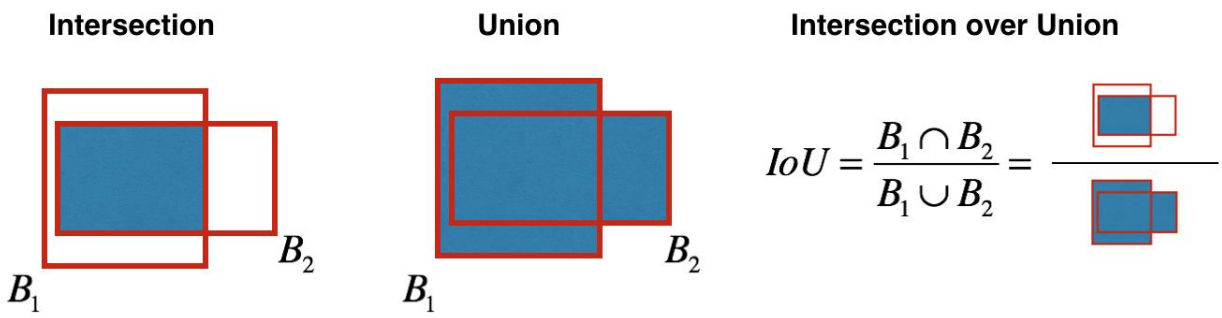

Рисунок 2.14 - IoU метрика

На початку можна виставити поріг для цієї метрики, і якщо передбачений bounding box буде вище цього порогу, то у нього буде objectness що дорівнює одиниці, а всі інші bounding boxes, у яких objectness нижче, будуть виключені. Ця величина objectness необхідна, для розрахунку загального confidence score (на скільки ми впевнені, що це саме потрібний нам об'єкт розташований всередині передбаченого прямокутника) у кожного певного об'єкта.

Коли на вхід подається картинка з датасету в YOLO, відбувається feature extraction на початку, а в кінці у нас виходить CNN шар, який розповідає нам про всіх клітинах, на які ми «розділили» нашу картинку. І якщо цей шар розповідає «неправду» про клітинки на зображенні, то ми отримуємо великий Loss, щоб потім його зменшувати при подачі в нейронну мережу наступних картинок. Даний процес зображено на рисунку 2.15[15]

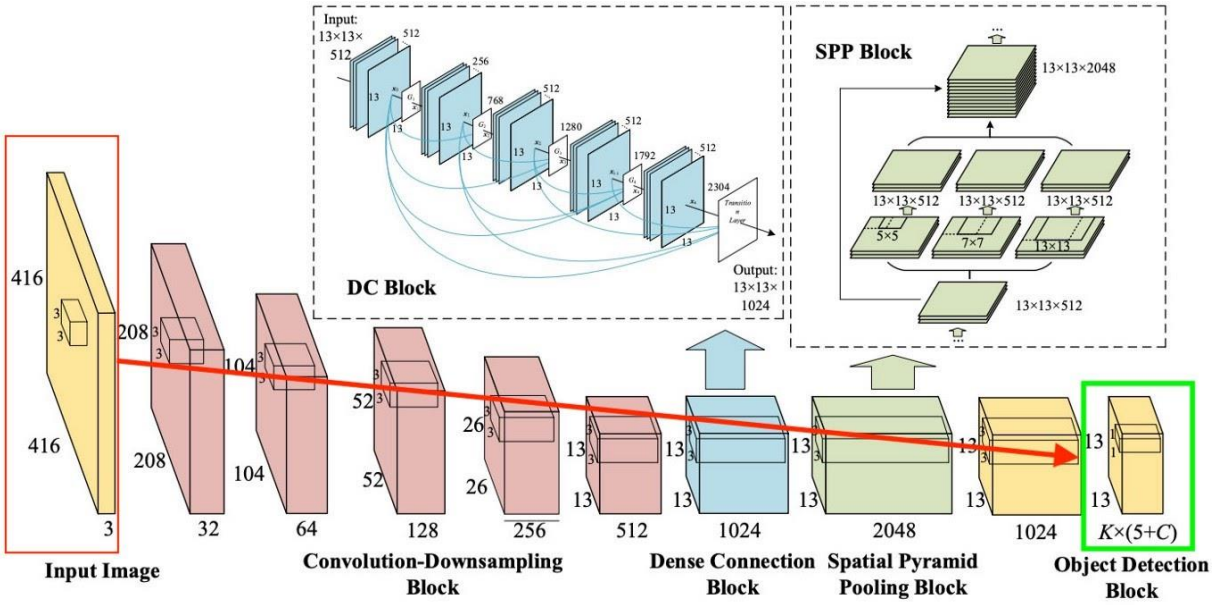

Рисунок 2.15 - YOLO архітектура

Як ми бачимо з рисунку, цей шар, розміром 13x13 (для картинок початкового розміру 416x416) для того, щоб розповідати про «кожну клітину» на зображенні. З цього останнього шару і дістається інформація, яку ми хочемо.

Для кожного anchor box YOLO прогнозує 5 параметрів:

$b_x=\sigma(t_x) + c_x$

$b_y=\sigma(t_y) + c_y$

$b_w=p_w e^{tw}$

$b_h=p_h e^{th}$

$Pr(object) * IoU(b, object) = \sigma(t_o)$

де t_x, t_y, t_w, t_h- передбачення YOLO, c_x, c_y - координата верхньої лівого кута необхідної grid cell, p_w, p_h - ширина та висота певного anchor box, b_x, b_y, b_w, b_h - параметри для передбаченого bounding box $\sigma(t_o)$ - confidence score. Ці параметри схематично зображено на рисунку 2.16.

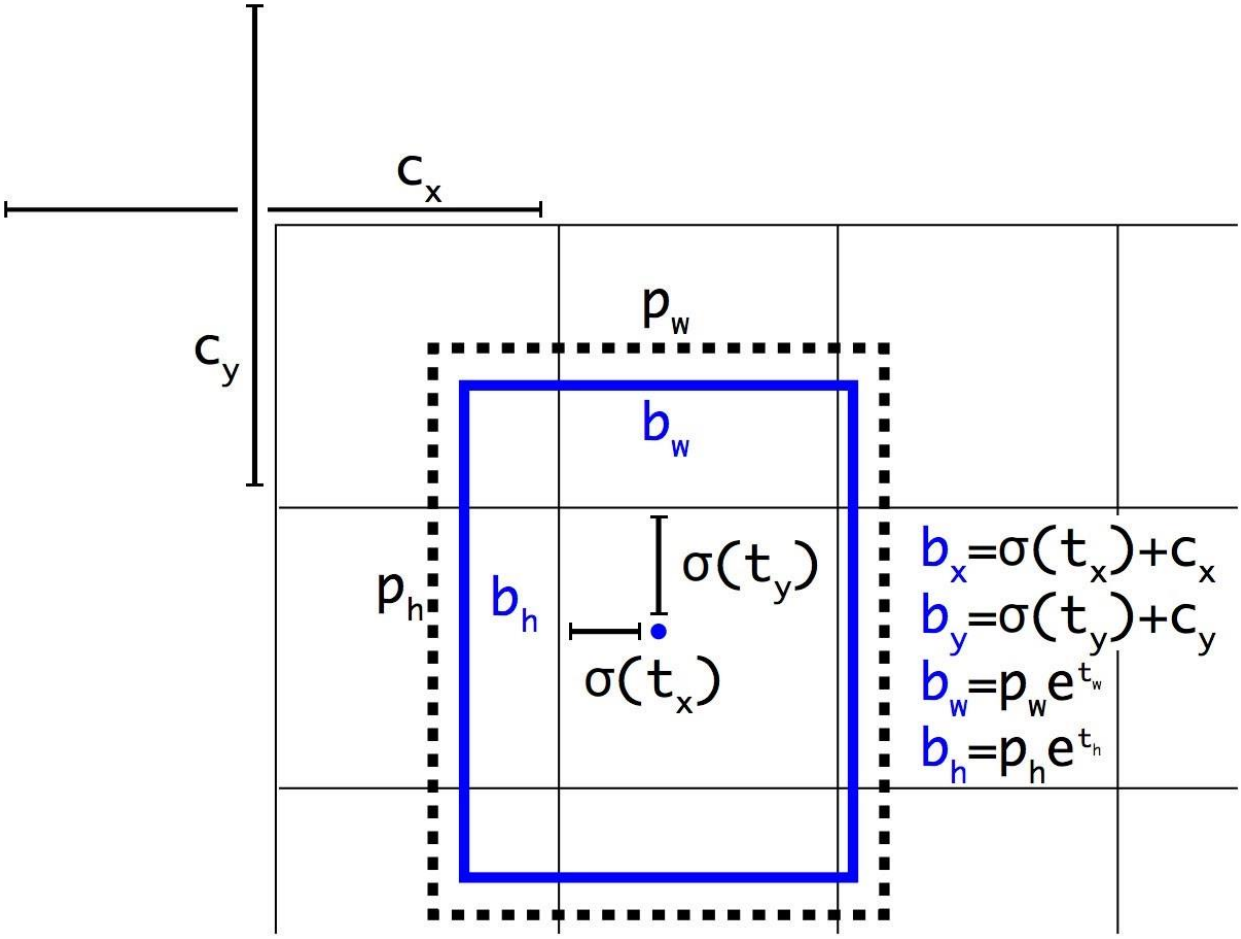

Рисунок 2.16 - Прогнозовані YOLO параметри

Як можна зрозуміти з цієї схеми, завдання YOLO - максимально точно передбачити ці параметри, щоб максимально точно визначати об'єкт на зображенні. А confidence score, який визначається для кожного передбаченого bounding box, є певним фільтром для того, щоб відсіяти зовсім неточні прогнози. Для кожного передбаченого bounding box ми множимо його IoU на ймовірність того, що це певний об'єкт (імовірнісний розподіл розраховується під час навчання нейронної мережі), беремо кращу ймовірність з усіх можливих, і якщо число після множення перевищує певний поріг, то ми можемо залишити цей передбачений bounding box на зображенні[16].

Далі, коли залишилися тільки передбачені bounding boxes з високим confidence score, наші передбачення (якщо їх візуалізувати) можуть виглядати так як на рисунку 2.17.

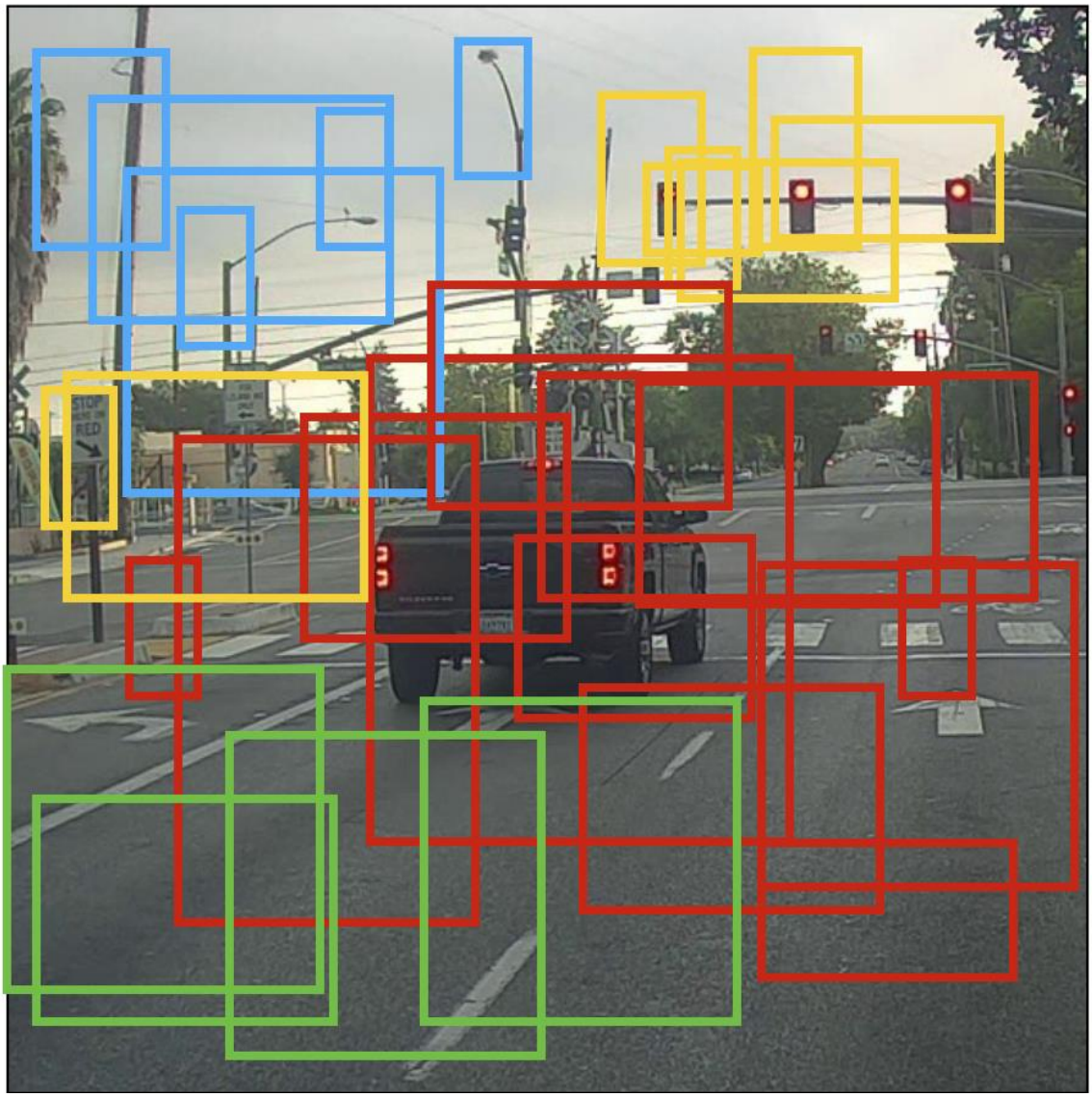

Рисунок 2.17 - Приклад візуалізації

Тепер, використовуючи техніку NMS (non-max suppression), можна відфільтрувати bounding boxes таким чином, щоб для одного об'єкта був тільки один передбачений bounding box, цей процес демонструється на рисунку 2.18.

Рисунок 2.18 - Приклад фільтрації bounding boxes

Оскільки метою даної роботи є побудова системи, що проводить детекцію в реальному часі було вирішено використовувати саме архітектуру YOLO, оскільки вона є швидшою в порівнянні з архітектурами сімейства R-CNN при цьому останн версії (3, 4) не поступаються в якості розпізнавання[16]. Порівняння YOLO з двоетапними детекторами наведено на рисунку 2.19.

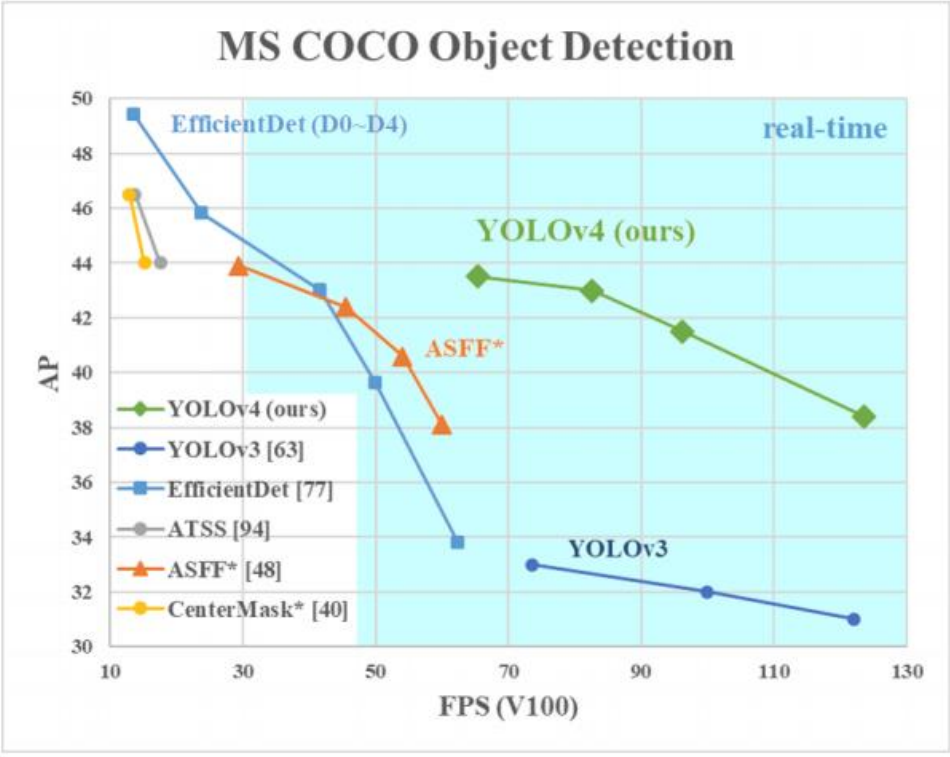

Рисунок 2.19 - Порівняння швидкості та якості розпізнавання різними мережами

2.7.1 DeepSORT

Оскільки YOLO сам по собі є лише детектором, а метою даної роботи є розпізнавання об’єктів в реальному часі, то необхідно використовувати додаткові алгоритми для того аби пов’язувати об’єкти з попереднього кадру з об’єктами на новому кадрі. Для цього використовується технологія під назвою DeepSORT.

Спочатку, за допомогою object detection, визначаються позиція, розмір і клас одного bounding box. Потім можна застосувати Угорський алгоритм, щоб зв'язати певні об'єкти з ID об'єктів, які раніше були на кадрі і відстежуються за допомогою фільтрів Калмана. Але технологія DeepSORT дозволяє поліпшити точність визначення і зменшити кількість перемикань між об'єктами, коли, припустимо, один чоловік на кадрі загороджує ненадовго іншого, і тепер людина, яку загородили, вважається новим об'єктом.

Це робиться додаванням так званого «зовнішнього вигляду» людей, які з'являються на кадрі (appearance). Цей зовнішній вигляд був натренований окремою нейронною мережею, яка була створена авторами DeepSORT. Вони використовували порядку 1,100,000 картинок з понад 1000 різними людьми, щоб нейронна мережа правильно передбачала В оригінальному SORT є проблема - так як там не використовується зовнішній вигляд об'єкта, то за фактом, коли об'єкт щось закриває на кілька кадрів (наприклад, інша людина або колона всередині будівлі), то алгоритм потім привласнює інший ID цій людині - в наслідок чого так звана «пам'ять» про об'єкти у оригінального SORT досить короткострокова.

Таким чином об'єкти мають дві властивості - їх динаміка руху та їх зовнішній вигляд. Для динаміки є показники, які фільтруються і передбачаються за допомогою фільтра Калмана - *(u, v, a, h, u ', v', a ', h')*, де *u, v* - це позиція передбаченого прямокутника по *X і Y, a* - це співвідношення сторін передбаченого прямокутника (aspect ratio), *h* - висота прямокутника, і похідні по

кожній величині. Для зовнішнього вигляду тренували нейронну мережу, яка мала структуру зображену на рисунку 2.20

Name	Patch Size/Stride	Output Size
Conv 1	$3 \times 3/1$	$32 \times 128 \times 64$
Conv 2	$3 \times 3/1$	$32 \times 128 \times 64$
Max Pool 3	$3 \times 3/2$	$32 \times 64 \times 32$
Residual 4	$3 \times 3/1$	$32 \times 64 \times 32$
Residual 5	$3 \times 3/1$	$32 \times 64 \times 32$
Residual 6	$3 \times 3/2$	$64 \times 32 \times 16$
Residual 7	$3 \times 3/1$	$64 \times 32 \times 16$
Residual 8	$3 \times 3/2$	$128 \times 16 \times 8$
Residual 9	$3 \times 3/1$	$128 \times 16 \times 8$
Dense 10		128
Batch and ℓ_2 normalization		128

Рисунок 2.20 - Архітектура НС для впізнавання зовнішнього вигляду об’єкту

Вона видавала feature vector, розміром 128x1. А далі, замість того, щоб вираховувати дистанцію між певними об'єктами за допомогою YOLO, і об'єктами, за якими ми вже спостерігали на кадрі, а потім приписувати певний ID просто за допомогою відстані Махалонобіса, автори створили нову метрику для підрахунку відстані, яка включає в себе як передбачення за допомогою фільтрів Калмана, так і коефіцієнт Отіа.

В результаті відстань від певного YOLO об'єкта до передбаченого фільтром Калмана об'єкта (або об'єкта, який вже є в числі тих, що спостерігався на попередніх кадрах) дорівнює:

$$D = Lambda * D_k + (1 - Lambda) * D_a \quad (6)$$

Де D_a - це дистанція по зовнішній схожості, а D_k – відстань Махалонобіса. Далі ця гібридна дистанція застосовується в Угорському алгоритмі, щоб якраз правильно впорядкувати певні об’єкти з наявними ID.

Висновок

В даному розділі було досліджено існуючі методи та підходи в сфері розпізнавання об’єктів. На основі порівняльного аналізу було обрано оптимальний алгоритм, а саме YOLO, через його швидкодію, що є важливою якістю для задачі розпізнавання в реальному часі. В той самий час, даний алгоритм суттєво не поступається двоетапним детекторам у якості розпізнавання, у сфері для якої планується його використання.

Також було досліджено алгоритм, що є необхідним для розпізнавання у відеопотоці (DeepSORT). Цей алгоритм виконує задачу поєднання інформації про об’єкти з попередніх кадрів з об’єктами на новому кадрі, що також позитивно відображається на якості та швидкості розпізнавання.

3 ВИБІР ТА НАВЧАННЯ МОДЕЛІ

3.1 Метрики якості

В задачах object detection найчастіше для оцінки коректності побудовки обмежуючої рамки використовують в якості метрики відношення площ обмежуючих рамок (Intersection over Union)

$$\text{IoU} = S(A|B)/S(A\&B) \tag{7}$$

A і B це відповідно передбачена та реальна обмежуючі рамки відповідно, таким чином чим вище значення цієї метрики (максимальне значення 1) тим краще співпадіння цих рамок. На основі цієї метрики також розроблена mAP (mean average precision). Ця метрика є усередненою точністю по всіх категоріях. В загальному вигляді середня точність це площа під precision-recall кривою.

$$\text{AP}=\int_0^1 p(r)dr \tag{8}$$

де p – точність

r – повнота з гіпотези, що обмежувальна рамка визначена вірно, якщо IoU 0.5. AP знаходиться в інтервалі від 0 до 1, як і точність з повнотою, а відповідно і mAP. На практиці середня точність часто рахується по точках, значення повноти яких рівномірно розподілено в діапазоні [0; 1]

$$\text{Apc} = 1/11 * (\text{AP}_c(0) + \text{AP}_c(0.1) + .. \text{AP}_c(1)) \tag{9}$$

Також дуже популярним є сет метрик COCO mAP. COCO — один з найбільших датасетів, що використовуються для навчання нейронних мереж. COCO mAP використовується інтерпольована за 101 точкою середня точність. AP в COCO рахується не для різної повноти, а для різного значення IoU.

AP@[.5:.95] відповідає середній точності розрахованій на IoU, що змінювалися в діапазоні від 0.5 до 0.95 з кроком 0.05. Також в COCO містяться й інші метрики, що наведені на рисунку 3.1. Для COPO mAP це те ж саме, що і AP.

```
Average Precision (AP):
   AP                  % AP at IoU=.50:.05:.95 (primary challenge metric)
   AP^IoU=.50          % AP at IoU=.50 (PASCAL VOC metric)
   AP^IoU=.75          % AP at IoU=.75 (strict metric)
AP Across Scales:
   AP^small            % AP for small objects: area < 32^2
   AP^medium           % AP for medium objects: 32^2 < area < 96^2
   AP^large            % AP for large objects: area > 96^2
Average Recall (AR):
   AR^max=1            % AR given 1 detection per image
   AR^max=10           % AR given 10 detections per image
   AR^max=100          % AR given 100 detections per image
AR Across Scales:
   AR^small            % AR for small objects: area < 32^2
   AR^medium           % AR for medium objects: 32^2 < area < 96^2
   AR^large            % AR for large objects: area > 96^2
```

Рисунок 3.1 - Метрики COCO

3.2 Вибір архітектури

Враховуючи те, що мережа, яка розробляється буде використовуватися в системі яка повинна розпізнавати об’єкти в реальному часі, необхідно забезпечити достатню швидкодію. Цьому критерію задовольняють одноетапні детектори серед яких найрозповсюдженими є YOLO детектори. Ці детектори вперше були представленні в 2016 році і на той момент значно програвали в якості розпізнавання Faster R-CNN, а виграш в швидкодії був доволі незначний, навесні 2020 року було представлено нову - 4 версію архітектури YOLO. В основі цього детектора є 2 частини backbone та head. Перша преднавчена на наборі ImageNet та використовується для детекції (в YOLO для цього було вирішено використовувати CSPDarknet53), а друга відповідно для класифікації об’єктів, що потрапили у рамку (YOLOv3 відповідно). Також, в той самий час, коли

зазвичай для покращення роботи згорткових мереж використовують так звану data augmentation. В YOLOv4 було для цього використано новий метод — Mosaic. Його суть полягає в тому, що поєднується 4 різних зображення з різним контекстом, це дозволяє проводити детекцію об’єктів поза їх звичним контекстом. Також batch normalization розраховує статистику активації з 4 різних зображень на кожному шарі, що дозволяє зменшити необхідність в використанні великих батчів.

Також розробниками була запропонована нова data augmentation техніка, що отримала назву Self-Adversarial Training (SAT). Ця техніка працює в 2 етапи. На першому етапі замість вагів мережі використовує початкове зображенн, що змушує мережу виконувати так звану «конкурентну атаку», це призводить до того, що на вихідному зображенні немає необхідного об’єкту. Після цього, на другому етапі мережа тренується знаходити об’єкт на модифікованому зображенні.

Згідно досліджень, така комбінація дозволила значно покращити результати відносно мереж, що вже існували. Їх порівняння наведено в таблиці 3.1.

Таблиця 3.1 - Порівняння моделей

Method	**Size**	**FPS**	**AP**	**AP_{50}**	**AP_{75}**	**APs**	**AP_m**	**AP_l**
YOLOv4	608	62	43.5%	65.7%	47.3%	26.7%	46.7%	53.3%
EfficientDet-D1	640	50.0	39.6%	58.6%	42.3%	17.9%	44.3%	56.0%
RFBNet	512	41.5	33.9%	54.3%	36.2%	14.7%	36.6%	50.5%
YOLOv3	608	20	33.0%	57.9%	34.4%	18.3%	35.4%	41.9%

З даних, що наведені в таблиці добре видно, що YOLOv4 має найбільшу швидкість розпізнавання в порівнянні з іншими мережами.Варто зазначити, що YOLOv3 не здатна забезпечити розпізнавання в реальному часі оскільки

зазвичай, роботою в реальному часі вважають розпізнавання більше 30 кадрів на секунду.

Також, заслуговує на увагу той факт, що середня точність розпізнавання малих об’єктів, яка є слабкою стороною для всіх детекторів досягла 26 відсотків, що перевищило показники інших мереж. Звичайно, показники можуть здаватися досить низькими, особливо зважаючи на точність якої досягли штучні нейронні мережі в задачах простої класифікації об’єктів на зображеннях, де точність може перевищувати 80%, але варто пам’ятати, що в задачах object detection великий вплив на значення метрик чинить коректність рамок та початкова розмітка, яка зазвичай проводиться вручну. Також об’єкти на зображеннях, що опинилися в регіонах інтересу можуть бути частково закритими іншими об’єктами або їх результуюче зображення буде в досить низькій роздільній здатності.

Для тренування нашої мережі було встановлено наступні параметри:

а) розмір батчу 64;

б) кількість класів 9;

в)розмір вхідного шару 416x416;

г) кількість каналів 3;

д) максимальна кількість ітерацій 18000;

е) кроків 14400, 16200;

є) моментум 0.949;

ж) кількість фільтрів в згорткових шарах 42.

3.3 Вибір даних для навчання моделі

Для навчання мережі, що буде використовуватися в системі, що розробляється було обрано KITTI-dataset цей датасет створений спеціально для розробників автономних автомобілів. Він складається з даних, що отримані з

стереопари камер, а також з лідару, що були встановлені на автомобілі який їздив вулицями Карлсруе. Загалом 15007 зображень в тестовій та навчальній вибірці.

Розподіл зображень наведено на зображені 3.2

Рисунок 3.2 - Розбиття датасету

Об’єкти що містяться в цьому сеті належать до 9 різних класів:

- автомобіль;
- вантажівка;
- пішохід;
- трамвай;
- велосипедист;
- мікроавтобус;
- людина, що просто сидить (винесені в окремий клас оскільки на відміну від пішоходів не можуть несподівано потрапити на дорогу);
- різне (об’єкті ни дорозі, що не відносяться до цих класів, наприклад причеп);
- не потребують уваги (для об’єктів, що належать до заданих класів, але знаходяться на великій відстані від автомобіля).

Нижче на рисунку 3.3 наведено зразок розміченного зображення.

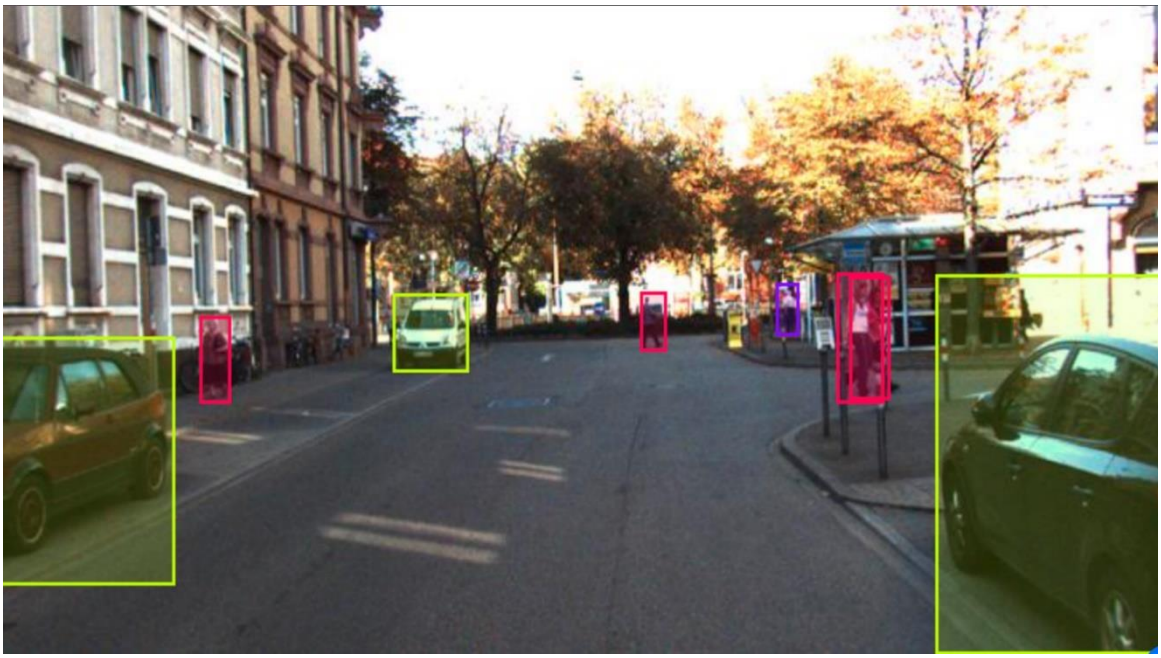

Рисунок 3.3 - Зразок розміченного зображення

Також на рисунку 3.4 наведено приклад файлу з розміткою класів на зображенні

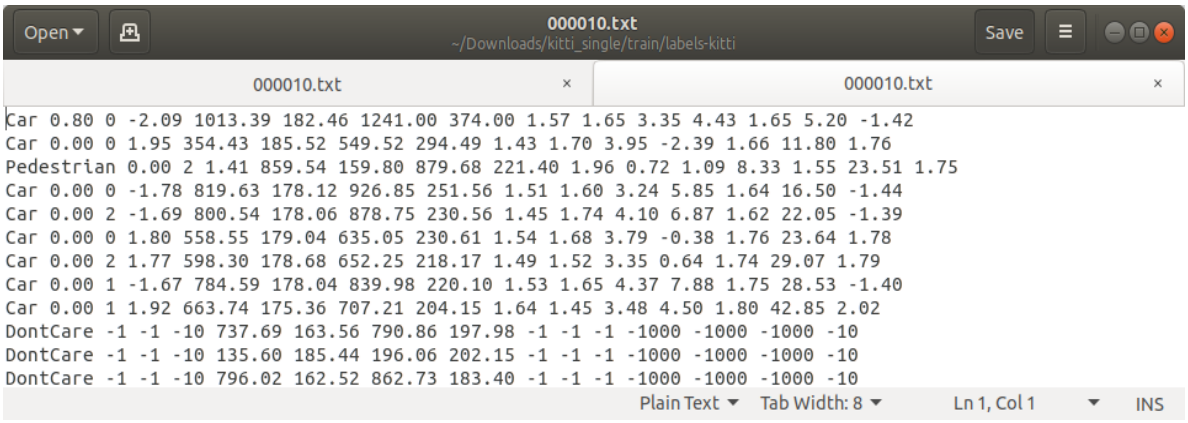

Рисунок 3.4 - Приклад файлу з розміткою класів на зображенні

Варто зазначити, що формат розмітки в KiTTI датасеті відрізняється від того, що прийнято для мережі YOLO. Тому було написано спеціальний адаптер, який привів розмітку до прийнятого вигляду. Скріншот перетвореної розмітки наведено нижче. Перше число відповідає номеру класу (список класів додаються в окремому файлі), друге та третє відповідають координатам центру рамки, четверте та п’яте відповідають її ширині та висоті. Адаптована розмітка наведена на зображенні 3.5.

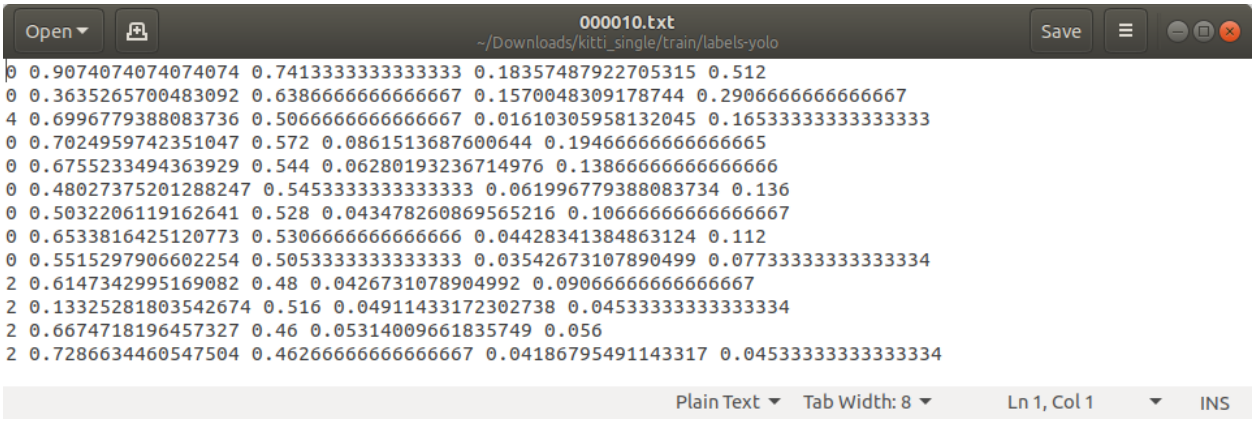

Рисунок 3.5 - Адаптована розмітка

Після виконання підготовчого етапу було проведено навчання моделі. Варто зазначити, що в якості початкових ваг було використано ваги мережі, що була преднавчена на COCO датасеті.

Приклади мозайок, побудованих системою під час навчання (після проходження через мережу) продемонстровано на рисунку 3.6

Рисунок 3.6 - Приклади мозайок

Графіки навчання зображено на графіках наведених нижче. На рисунку 3.7 зображено графік зміни mAP в процесі навчання

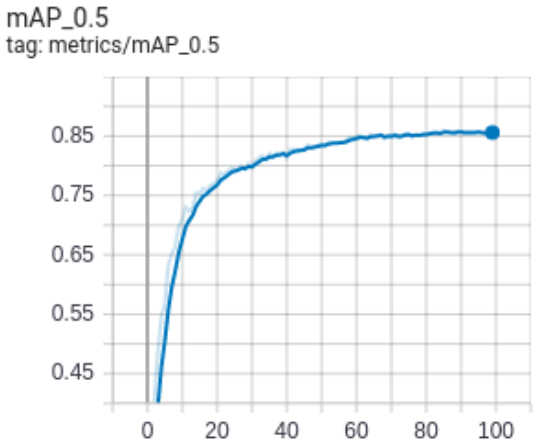

Рисунок 3.7 - Графік зміни mAP в процесі навчання

Як можна помітити, підхід з донавчанням нашої мережі на основі преднавчених ваг виправдав себе і для значення IoU = 0.5 під час навчання вдалося досягти середньої точності в 85.4% На наступному рисунку 3.8 зображено інтервальне mAP для IoU від 0.5 до 0.95

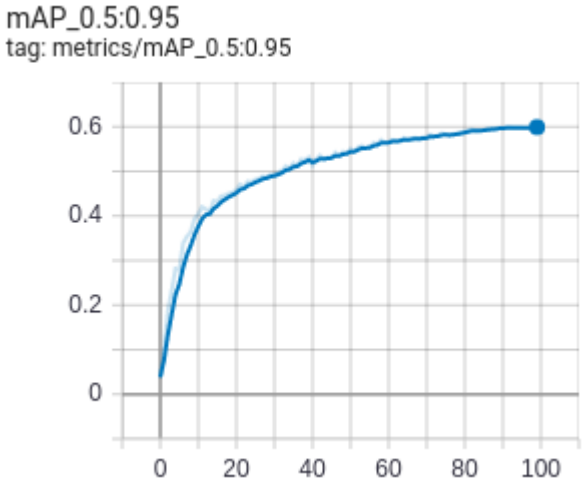

Рисунок 3.8 - mAP для IoU від 0.5 до 0.95

На цьому графіку помітно, що якщо оцінювати середню точність для IoU в діапазоні, то результати є гіршими, очевидно, це спричинено тим, досягти високої точності співпадіння побудованих мережею рамок з істинними все ж не вдається, але це не є критичною проблемою, оскільки даний факт може бути спричинений і досить грубою розміткою початкового датасету. Тепер

розглянемо результати саме класифікатора мережі. Графіки точності та повноти яких було досягнуто зображено на рисунках 3.9 та 3.10 відповідно.

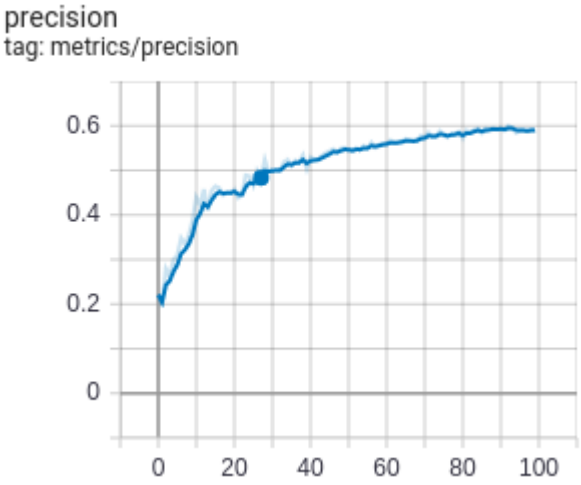

Рисунок 3.9 - Графік точності

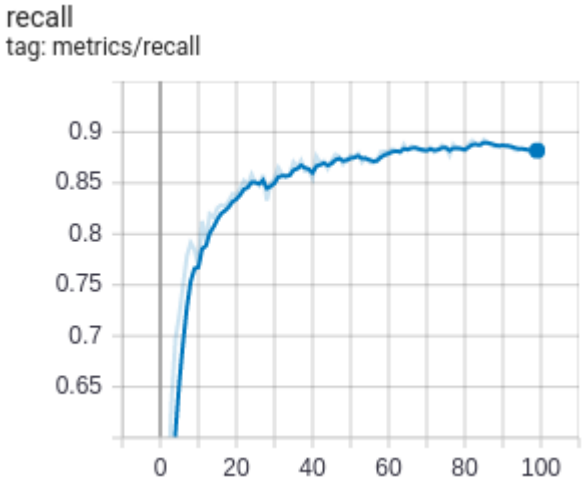

Рисунок 3.10 - Графік повноти

Враховуючи контекст для якого будується система, повнота є більш важливою характеристикою за точність, оскільки для систем безпеки автомобіля більш важливо помітити автомобіль взагалі, ніж визначити його тип, до того ж високої точності важче для даної сфери через перекриття об’єктів, відстань до них та ракурси. До того ж є класи, які подібні до інших, але зустрічаються рідше. Наприклад мікроавтобус (наприклад невеликі T3 від Volkswagen) та позашляховик очевидно відносно схожі формою кузову, але в той самий час належать до різних класів. Приклад такої помилкової класифікації наведено

нижче на рисунку 3.11. В той самий час повнота розпізнавання об’єктів є досить високою, що відповідає вимогам які висуваються до подібних систем.

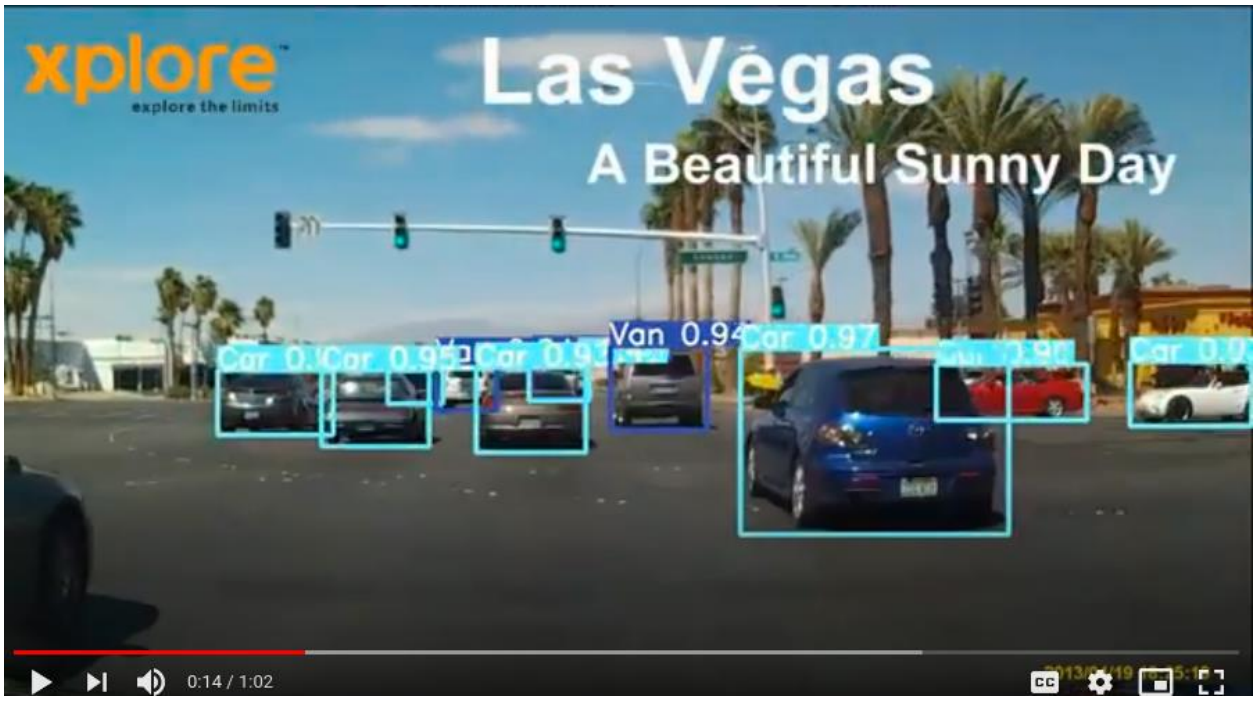

Рисунок 3.11 - Розпізнавання з відеоряду

Як видно, один з автомобілів попереду було віднесено до іншого класу, хоча в аналогічних умовах навіть людині досить важко класифікувати всі об’єкти вірно. Варто відзначити, що зображення являє собою скріншотом з тестового відео на якому було продемонстровано можливість побудованої системи проводити розпізнавання в реальному часі.

На наступному зображенні (рисунок 3.12) продемонстровано розпізнавання об’єктів на звичайному зображенні з тестової вибірки, видно, що за відносно сприятливих умов система розпізнала 2 автомобілі з досить високою точністю, що свідчить про гарні результати, яких вдалося досягти в хоі навчання

Рисунок 3.12 - Розпізнавання з зображення

На наступних графках (рисунок 3.13 та 3.14 показано зміну рівня помилки для рамок та классифікації, як видно мережа продемонструвала гарну збіжність.

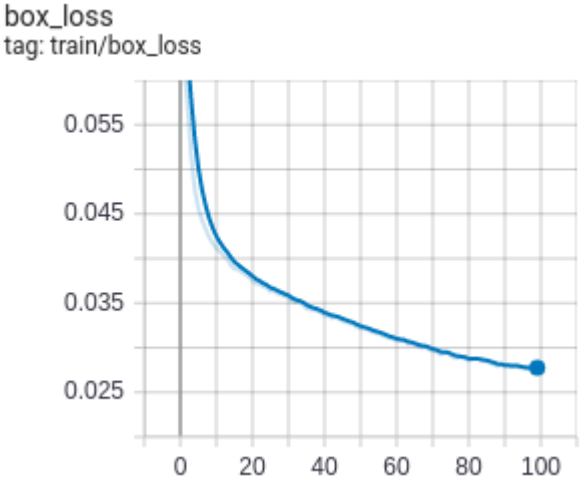

Рисунок 3.13 - Рівень помилки для обмежуючих рамок

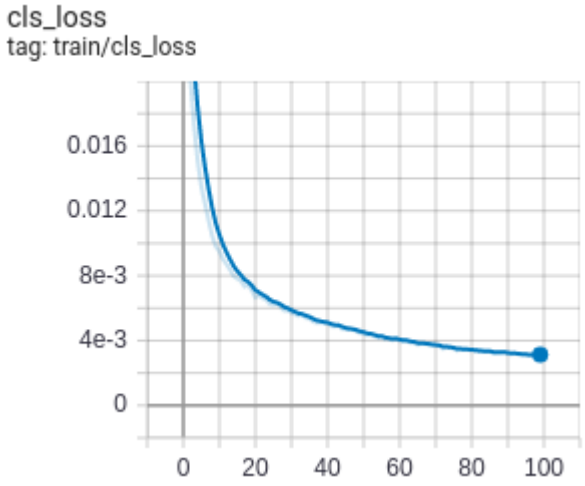

Рисунок 3.14 - Рівень помилки для класифікатора

Висновок

В данному розділі було проведенно порівняння обраної архітектури з іншими, описано датасет та процес навчання, проаналізовано результати. Також було наочно продемонстровано розпізнавання об’єктів у реальному часі, а також підтверджено високі характеристики натернованої мережі.

4 ОПИС ПРОГРАМНОГО ПРОДУКТУ

4.1 Вибір платформи та загальний опис ПП

Оскільки навчання згорткових нейронних мереж є вимогливим до ресурсів процесом, а також потребує графічного процесору, якого в моєму ноутбуку не було, тому для навчання було використано сервіс Google Colab в якому користувачеві з певними обмеженнями надається обчислювальна потужність NVIDIA Tesla K80.

Для розробки програмного продукту було використано bash, Python, PyTorch, конфігурація мережі описується в конфігураційному файлі. Керування системою здійснюється за допомогою скриптів Python.

Сама по собі мережа YOLOv4 була розроблена з використанням Darknet фреймворку. Darknet - це фреймворк з відкритим кодом, написана на мовах C та CUDA. Він швидкий, простий у встановленні та підтримує обчислення на центральному процесорі та графічномі процесорі, що надае можливість значно пришвидшити процес навчання. Код цього фреймворку є відкритим та його можна знайти в репозиторії GitHub. Для полегшення взаємодії з нейронною мережою було використано відомий фреймворк PyTorch.

PyTorch - це пакет для наукових обчислень на основі Python, який використовує потужність графічних процесорів. Це також одна з кращих платформ для дослідженнь в сфері глибокого навчання, створена для забезпечення максимальної гнучкості і швидкості. Він відомий тим, що надає дві функції найвищого рівня; а саме тензорні обчислення з потужною підтримкою прискорення графічного процесора і побудова глибоких нейронних мереж на стрічкових системах автоградації.

Існує безліч бібліотек Python, які можуть змінити спосіб виконання глибокого навчання і штучного інтелекту, і це одна з таких бібліотек. Одна з ключових причин успіху PyTorch полягає в тому, що він повністю є сумісним із

Python та дозволяє легко створювати моделі нейронних мереж. Хоча це все ще досить новий продукт в порівнянні з конкурентами, однак він швидко розвивається та постійно оновлюється.

Основними задачами програмного продукту були:

а) реалізація спрощеної взаємодії з даними на яких проводиться навчання, для цього використано сервіс RoboFlow, який дозволяє зручно переглядати тренувальні дані, а також у разі необхідності додавати нові або змінювати розбиття на тренувальну та валідаційну вибріки.

б) конвертація данних в YOLO формат;

в) реалізація збереження навчених ваг для використання в окремих застосунках;

г) написання інструкції з використання даного продукту.

4.2 Опис інтерфейсу програми

На даний момент програма не містить графічного інтерфейсу. Взаємодія користувача із програмним продуктом відбувається за допомогою linux консолі. На рисунку 4.1 зображено директорію проекту.

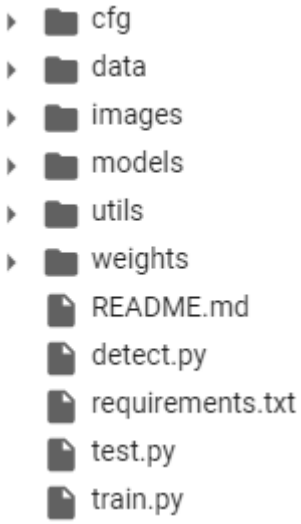

Рисунок 4.1 - Директорія проекту

В директорій **cfg** знаходяться конфігуранційні файли системи, в яких описуються параметри нашої мережі, які передаються скрипту train.py в якості параметру.

В директорії data знаходяться yaml та txt файли, що описують датасет. Зокрема файл data.yaml містить список класів в датасеті та адресу за якими розташовані тренуальні та тестові вибірки. Приклад файлу зображено на рисунку 4.2

data.yaml

```
train: ../train/images
val: ../valid/images
test ../test/images

nc: 9
names: ['Car', 'Cyclist', 'DontCare', 'Misc', 'Pedestrian', 'Person_sitting', 'Tram', 'Truck', 'Van']
```

Рисунок 4.2 - Файл з описом датасету

Дані можна імпортувати за допомогою завантаження архіву датасету вручну, або використовуючи утиліту wget, яка завантажує дані розташовані за певною інтернет адресою. Приклад її використання наведено на рисунку 4.3

```
!curl -L "https://app.roboflow.com/ds/nimN8ciE5y?key=y5DfxPZNMI" > roboflow.zip; unzip roboflow.zip; rm roboflow.zip
extracting: valid/labels/005978_jpg.rf.89411f76a0d60ae2ce143980013c3d01.txt
extracting: valid/labels/005988_jpg.rf.961d7f8107c46392f68e5ad3629c4cfb.txt
extracting: valid/labels/006005_jpg.rf.6f8cdf265d5ec499815858164b83af18.txt
extracting: valid/labels/006015_jpg.rf.968c53ae7f5a8f1ddc7a560dbf75c1f4.txt
extracting: valid/labels/006017_jpg.rf.766c797aefbfe4b8f6361b2af3c4a0f7.txt
extracting: valid/labels/006033_jpg.rf.a88f0a618db8b7dac524cd9afc808aa9.txt
extracting: valid/labels/006034_jpg.rf.c73a450a8775cd98bddf5c586dd030f7.txt
extracting: valid/labels/006042_jpg.rf.4bf4ccb31910618724e306182157199f.txt
extracting: valid/labels/006047_jpg.rf.06f84e00736e9e64d7a3329b049d171c.txt
extracting: valid/labels/006061_jpg.rf.ab92794eaf090066f70322b1d9d6210b.txt
extracting: valid/labels/006065_jpg.rf.020a2477dbeac512e51aad829a6cbbf0.txt
extracting: valid/labels/006071_jpg.rf.8ea453a3e5c6ae6b62215e78340cd352.txt
extracting: valid/labels/006077_jpg.rf.4b233757de1df94e7b7d182c62a6d497.txt
```

Рисунок 4.3 - Завантаження даних

В директорії weights зберігаються ваги моделі для використання преднавчених мереж. Хоча їх використання не є обов’язковим, але даний підхід дозволяє покращити якість навчання, оскільки зазвичай моделі преднавчаються на досить великих датасетах (ImageNet та COCO).

Директорії utils та models використовуються для зберігання безпосередньо PyTorch імплементації мережі та не потребують редагування користувачем для проведення навчання або тестування мережі.

kitti_to_yolo.py - даний скрипт використовується для конвертації даних в yolo формат, в якості параметрів він приймає адресу директорії із зображеннями, а також список класів які містяться в датасеті він не є обов'язковим, але в ході написання магістерської дисертації його було створено для генерації даних для навчання.

yolov4.cfg містить конфігурацію мережі, а саме параметри згорткових шарів, кількість каналів (3 для кольорових зображень), розмір батчів (64), max_batche - максимальна кількість батчів, розраховується як кількість класів * 2000, **filters** — встановлюється як (кількість класів + 5) * 3 також в цьому файлі вказуються розміри вхідних зображень, більший розмір зображень призводить до кращої точності мережі, але знижує швидкість навчання та розпізнавання.

train.py — скрипт, що запускає тренування мережі, приймає на вхід конфігураційний файл, а також в якості опціонального параметру ваги преднавченої мережі. Після завершення виконання даного скрипту, в директорії проекту з'являються файли з вагами мережі, що показала найкращі результати на валідаційній вибірці, які можливо використати для оцінки якості роботи моделі та зберегти для проведення розпізнавання на зображеннях або відео. На рисунку 4.4 зображено приклад запуску скрипту.

```
1 %%time
2 %cd /content/yolov4/
3 !python train.py --batch 64 --epochs 100 --data '../data.yaml' --cfg ./models/yolov4.conf --weights yolov4.pt --name yolov4_results
```

Рисунок 4.4 - Приклад заппуску скрипту

test.py — скрипт, що в якості параметрів приймає адресу файлу з вагами, що були отримані в ході навчання моделі, а також директорію в якій знаходяться

зображення з тестової вибірки або відеофайл для тестування швидкості розпізнавання.

detect.py – скрипт який безпосередньо використовується для розпізнавання за допомогою вже навченої мережі. Цей скрипт отримує в якості параметрів відео або зображення на якому буде проходити розпізнавання, а також ваги, які будуть використовуватися для детекції. Також вказується пороговий рівень «впевненості». Об'єкти щодо яких впевненість мережі нижча за вказану не будуть відмічатися на результуючому зображені чи відео. Приклад активації скрипту наведено на рисунку 4.5

```
%cd /content/yolov4/
!python detect.py --weights ./weights/best.pt --conf 0.6 --source /content/gdrive/MyDrive/test3.mp4

/content/yolov4
Namespace(agnostic_nms=False, augment=False, classes=None, conf_thres=0.3, device='', exist_ok=False, i
Using torch 1.7.0+cu101 CUDA:0 (Tesla T4, 15079MB)

Fusing layers...
Model Summary: 232 layers, 7268094 parameters, 0 gradients
video 1/1 (1/1871) /content/gdrive/MyDrive/test3.mp4: 384x640 Done. (0.016s)
video 1/1 (2/1871) /content/gdrive/MyDrive/test3.mp4: 384x640 Done. (0.012s)
video 1/1 (3/1871) /content/gdrive/MyDrive/test3.mp4: 384x640 Done. (0.010s)
video 1/1 (4/1871) /content/gdrive/MyDrive/test3.mp4: 384x640 Done. (0.009s)
video 1/1 (5/1871) /content/gdrive/MyDrive/test3.mp4: 384x640 Done. (0.009s)
video 1/1 (6/1871) /content/gdrive/MyDrive/test3.mp4: 384x640 Done. (0.009s)
```

Рисунок 4.5 – Проведення розпізнавання

Висновки

В даному розділі було описано архітектуру програмного застосунку та описано інструменти, що використовувалися в процесі його створення. В подальшому, є можливість створення графічного інтерфейсу для данного застосунку, що повинно значно покращити досвід взаємодії користувача з програмою.

5 РОЗРОБКА СТАРТАП-ПРОЕКТУ

5.1 Опис ідеї проекту

Метою магістерської дисертації є підвищення безпеки кермування автомобілем за рахунок побудови системи розпізнавання об’єктів за допомогою вбудованих камер. Зміст ідеї, потенційні напрямки застосування та головні вигоди, що може отримати користувач товару представлено у таблиці 5.1

Таблиця 5.1 – Опис ідеї стартап-проекту

Зміст ідеї	Напрямки застосування	Вигоди для користувача
Додавання системи розпізнавання об’єктів у автомобільні прошивки	Побудова систем безпеки в сучасних автомобілях	Підвищення рівня безпеки на дорогах загального користування. Поліпшення зручності керування автомобілем
	Використання системи для модернізації існуючих автомобілів	Підвищення якості роботи та безпеки керування без великих вкладень

Запропонований метод відрізняється від існуючих великою швидкодією та відносно невеликою вартістю. Було визначено перелік техніко – економічних властивостей та характеристик ідей. Наступним етапом проведено аналіз потенційно можливих техніко-економічних властивостей та характеристик ідеї та існуючих аналогів. Для проведення такого аналізу, визначено попереднє коло опонентів, що представлені на ринку. Результати досліджень представлено у таблиці 5.2.

Таблиця 5.2 – Визначення сильних, слабких та нейтральних характеристик ідеї проекту

№	Хар-ка	Назви продуктів			Слабка сторона	Ней-на сторона	Сильна сторона
		Мій проект	NVIDIA DRIVE	NXP BlueBox			
1	Вартість	Неможливо обрахувати, залежить від моделі камери	Прблизно 15000 долларів	Приблизно 12000 доларів			Вартість значно менша
2	Швидкодія (fps)	Теоретично до 150 кадрів на секунду	2000	200	Швидодія нижча, але при руху з нормальною швидкістю це не критично		
3	Датчики	Камера	Камера	Лідар та камера		Менша кількість датчиків знижує вартість	

Продовження таблиці 5.2

№	Хар-ка	Назви продуктів			Слабка сторона	Ней-на сторона	Сильна сторона
		Мій проект	NVIDIA DRIVE	NXP BlueBox			
4	Можливість роботи в неспритливих умовах	Є, при використанні ІЧ камери	Є, при зниженні точності	Є при зниженні точності			Використання ІЧ камери в нічний час допомагає уникнути зниження точності

5.2 Технологічний аудит ідеї проекту

У таблиці 5.3 приведено показники технічного аудиту ідеї проекту, та визначено за якою технологією буде виготовлено продукт, проведено аналіз існування необхідних технологій та їх доступність.

Таблиця 5.3 – Технологічна здійсненність ідеї проекту

№	Ідея проекту	Технології її реалізації	Наявність технології	Доступність технології
1	Отримання зображення навколишніх об’єктів	Використання лідару	Наявні готові зразки	Доступна

Продовження таблиці 5.3

№	Ідея проекту	Технології її реалізації	Наявність технології	Доступність технології
2		Використання радару	Наявні готові зразки	Доступна
3		Використання камер	Наявні готові зразки	Доступна
4	Розпізнавання об’єктів на основі даних з датчиків	R-CNN	Наявна, але низька швидкість	Доступна
5		Faster R-CNN	Наявна у вільному доступі	Доступна
6		YOLO	Наявна у вільному доступі	Доступна

Обрана технологія реалізації проекту: отримання зображення з камери та детекція об’єктів за допомогою YOLO

5.3 Аналіз ринкових можливостей запуску стартап-проекту

Визначено ринкові можливості, які можна використати під час ринкового впровадження проекту та ринкових загроз, які можуть зашкодити реалізації

проекту. В таблиці 5.4 показано результати аналізу попиту та стану справ на ринку.

Таблиця 5.4 - Результати аналізу попиту та стану справ на ринку

№	Показники стану ринку	Характеристика
1	Кількість головних гравців, од	До 15
2	Загальний обсяг продаж, грн	345000
3	Динаміка ринку (якісна оцінка)	Зростає
4	Наявність обмежень для входу (вказати характер обмежень)	Необхідність сертифікації
5	Специфічні вимоги до стандартизації та сертифікації	Проходження всіх тестів безпеки, проведення великої кількості годин тестів на спеціальних полігонах і відокремлених дорогах
6	Середня норма рентабельності в галузі (або по ринку), %	73

Роблячи аналіз отриманих результатів, можна сказати, що ринок є доволі привабливим для входження.

Наступним етапом є визначення потенційних груп клієнтів, їх характеристики та формування орієнтовного переліку вимог до товару для кожної групи. Результати досліджень представлено у таблиці 5.5.

Таблиця 5.5 – Характеристика потенційних клієнтів стартап-проекту

№ п/п	Потреба, щоформує ринок	Цільовааудиторія (цільові сегменти ринку)	Відмінності уповедінці різнихпотенційнихцільових групклієнтів	Вимоги споживачів до товару
1.		Користувачі яких цікавлять сучасні тренди автомобільної індустрії	Бажання завжди користуватися останніми технологіями	Справедливаа ціна, високі показники якості
2.		Автоконцерни	Збільшення попиту на свою продукцію	Можливість інтеграції в уже куплений автомобіль, безпечність

Далі, проводимо аналіз ринкового середовища та складаємо таблиці факторів, що сприяють ринковому впровадженню проекта, та факторів, що йому перешкоджають. Результати представлено у таблиці 5.6 та 5.7.

Таблиця 5.6 – Фактори загроз

№ п/п	Фактор	Зміст загрози	Можлива реакція компанії
1.	Поява більш швидких методів	Можлива поява та велика популярність нового методу, що матиме кращі характеристики	Пошук способів якість системи

Продовження таблиці 5.6

№ п/п	Фактор	Зміст загрози	Можлива реакція компанії
2.	Відсутність інвестицій	За відсутності інвестицій неможливий початок виробництва продукту	Пошук джерел інвестицій
3.	Не проходження сертифікації	Без проходження сертифікації системи, неможлива її імплементація	Вдосконалення системи та Повторне проходження сертифікації

Таблиця 5.7 – Фактори можливостей

№ п/п	Фактор	Зміст можливості	Можлива реакція компанії
1.	Сертифікація та стандартизація	Вдало пройти сертифікаційну комісію та стандартизація	Вихід на ринок та доступ до інвестицій
2.	Отримання контракту	Підписання контракту з великим автовиробником	Збільшення обсягів виробництва
3.	Інвестиції	Отримання коштів	Вкладення інвестицій у нові технології та розширення функціоналу

Наступним етапом проведемо аналіз пропозиції та визначаємо рівень конкуренції. Результати аналізу представлено у таблиці 5.8

Таблиця 5.8 – Ступеневий аналіз конкуренції на ринку

№	Особливості Конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)
1	Чиста конкуренція	Майже всі автомобільні корпорації мають напрацювання в області систем безпеки	Розвиток технологічності, використання переваги у гнучності компанії для швидшого оновлення системи
2	Міжнародна конкуренція	Автомобільні компанії розташовані в різних країнах	Налагодження ланцюгів доставки в різні країни
3	Міжгалузева конкуренція	Розпізнавання об’єктів застосовується не лише в автоіндустрії	Можливість вести бізнес в ширшому діапазоні галузей
4.	Товарно-родова конкуренція	Створення товарів однієї категорії	Ведення активної Рекламної кампанії
5.	Конкурентні перевани нецінові	Покупці готові платити більше за більш якісну систему	Спрямування більшої частини доходів на розвиток технологій
6.	Марочна конкуренція	Система призначена для спільної цільової аудиторії	Введення активної маркетингової діяльності

Аналіз конкуренції в галузі за моделлю 5 сил М. Портера приведено у таблиці 5.9.

Таблиця 5.9 – Аналіз конкуренції в галузі за М. Портером

Складові аналізу	Прямі конкуренти в галузі	Потенційні конкуренти	Постачальники	Клієнти	Товари-замінники
	NVIDIA DRIVE, NXP BlueBox	Немає	Немає	Немає	Немає
Висновки	Конкурентна боротьба досить висока, через сильну виробничу та наукову базу конкурентів	Наявна можливість виходу на ринок після проведення сертифікації та стандартизації	Ні	Ні	Немає

З огляду на конкуренцію, робота на ринку можлива, проте доведеться конкурувати з гігантами технічної сфери, які довгий час знаходяться на ринку.

Визначимо фактори конкурентоспроможності, їх наведено у таблиці 5.10.

Таблиця 5.10 – Обґрунтування факторів конкурентоспроможності

№ п/п	Фактор конкурентоспроможності	Обґрунтування
1.	Використання різних типів сенсорів	Представлена система використовує кольорові камери та має можливість використання ІЧ камери,
2.	Низька собівартість	Використання лише звичайної камери, суттєво знижують витрати на комплектуючі

За визначеними факторами конкурентоспроможності проводимо аналіз сильних та слабких сторін стартап-проекту. Порівняльний аналіз представлено у таблиці 5.11

Таблиця 5.11 – Порівняльний аналіз сильних та слабких сторін проекту

№ п/п	Фактор конкурентоспроможності	Бали 1-20	Рейтинг товарів-конкурентів у порівнянні з власним проектом						
			-3	-2	-1	0	+1	+2	+3
1.	Комбінування сенсорів	10			+				
2.	Низька вартість	20	+						
3.	Можливість роботи вночі	5						+	
4.	Можливість роботи за поганих погодних умов	5						+	

Завершуючим етапом ринкового аналізу можливостей впровадження проекту являється SWOT – аналіз, на основі виділених ринкових загроз та можливостей, сильних та слабких сторін. SWOT – аналіз представлено у таблиці 5.12.

Таблиця 5.12 – SWOT – аналіз стартап-проекту

Сильні сторони: досить висока точність, низька ціна, висока швидкодія	Слабкі сторони: відсутність фізичних датчиків, неможливість сертифікації
Побудова реальної системи	Значна конкуренція з боку більш великих компаній

На основі SWOT – аналізу розроблено альтернативи ринкової поведінки для виведення стартап – проекту на ринок та орієнтовний час їх ринкової реалізації. Розроблені альтернативи показано у таблиці 5.13

Таблиця 5.13 – Альтернативи ринкового впровадження стартап-проекту

№ п.п	Альтернатива (орієнтовний комплекс заходів) ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
1.	Створення фізичної системи	Малоймовірно	19 місяців
2.	Вихід на ринок в якості потенційного проекту	Ймовірно	2 місяці
3	Підписання контракту з великим виробником	Малоймовірно	3

Аналізуючи наявні перспективи найвигіднішим варіантом є реалізація продукту не у вигляді готового продукту, а у вигляді софтверної частини, що проводить розпізнавання.

5.4 Розроблення ринкової стратегії проекту

На першому кроці визначимо стратегії охоплення ринку. Опис цільових груп представлено у таблиці 5.14

Таблиця 5.14 – Вибір цільових груп потенційних споживачів

№ п/п	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи (сегменту)	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
1.	Автоконцерни	Низька	20%	Висока	Низька
2.	Автовласники	Висока	45%	Низька	Висока
3.	Компанії, що займаються тюнингом автомобілів	Висока	80%	Низька	Середня
4.	Наукові центри	Низька	15%	Висока	Низька

Для роботи в обраному сегменті ринку сформулюємо базову стратегію розвитку. Визначення базової стратегії розвитку представлено в таблиці 5.15

Таблиця 5.15 – Визначення базової стратегії розвитку

№ п/п	Обрана альтернатива розвитку проекту	Ключові конкурентоспроможні позиції відповідно до обраної альтернативи	Базова стратегія розвитку*
1.	Вихід на ринок В якості софтверного продукту	Можливість виходу на ринок без створення фізичного готового продукту	Стратегія диференціації

У таблиці 5.16 представлено визначення базової стратегії конкурентної поведінки

Таблиця 5.16 – Визначення базової стратегії конкурентної поведінки

№ п/п	Чи є проект «першопрохідцем» на ринку?	Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?	Чи буде компанія копіювати основні характеристики товару конкурента, і які?	Стратегія конкурентної поведінки*
1.	Ні	Шукати нових	За потреби	Стратегія заняття конкурентної ніші

Визначення стратегії позиціювання представлено у таблиці 5.17

Таблиця 5.17 – Визначення стратегії позиціювання

№ п/п	Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкурентоспроможні позиції власного стартап-проекту	Вибір асоціацій, які мають сформувати комплексну позицію власного проекту (три ключових)
1.	Висока точність, достатня швидкодія, а також можливість реалізації на базі існуючих автомобілів	Стратегія диференціації	Розширення доступних до використання датчиків. Низька собівартість.	Низька собівартість, адаптивність, висока швидкодія.

5.5 Розроблення маркетингової програми стартап проекту

Сформуємо маркетингову концепцію продукту. Визначення ключових переваг концепції потенційного продукту представлено в таблиці 5.18.

Таблиця 5.18 – Визначення ключових переваг концепції потенційного товару

№ п/п	Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити)
1.	Швидкість розпізнавання об’єктів	Система надає можливість розпізнавання в реальному часі	Необхідно додатково підвищити швидкість розпізнавання для отримання переваги
2.	Низька ціна системи	Завдяки використаним алгоритмам розпізнавання можливість проводити детекцію лише завдяки камерам	Більш низька вартість зумовлена відсутністю необхідності використання дорогих сенсорів
3.	Можливість використання в якості модифікації	Система надає можливість встановлення на вже наявні автомобілі	Легкість встановлення та легкість налаштування системи

На наступному кроці визначимо цінові межі, якими необхідно керуватися при встановленні ціни на потенційний товар. Визначення меж встановлення ціни показано у таблиці 5.19.

Таблиця 5.19 – Визначення меж встановлення ціни

№ п/п	Рівень цін на товари замінники	Рівень цін на товарианалоги	Рівень доходів цільової групи споживачів	Верхня та нижня межі встановлення ціни на товар/послугу
1	Відсутні	16000 – 21000 $	3000 $/місяць	2 000 – 5 000 $

Далі. Формування системи збуту – Таблиця 5.20.

Таблиця 5.20 – Формування системи збуту

№ п/п	Специфіка закупівельної поведінки цільових клієнтів	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту
1.	Клієнт замовляє продукт, виготовляється система та доставляється клієнту	Встановлення продукту на автомобілі замовника	Один рівень	Комбінована

Розробимо концепцію маркетингової комунікації. Її розробку представлено у таблиці 5.21

Таблиця 5.21 – Концепція маркетингових комунікацій

№ п/п	Специфіка поведінки цільових клієнтів	Канали комунікацій	Ключові позиції, обрані для позиціонування	Завдання рекламного повідомлен ня	Концепція рекламного звернення
1.	Люди, що цікавляться сучасними технологіями	Інтернет	Низька ціна, висока якість, легкість встановлення	Демонстрація конкурентних переваг продукту	Інформаційні відео

Висновок

В даному розділі проведено підготовчий аналіз для впровадження розробленої системи в якості стартап проекту. Досліджено аналогічні конкурентні системи, встановлено сильні та слабкі сторони системи в порівнянні з ними.

Також було досліджено можливі шляхи розповсюдження продукту та його ймовірну аудиторію, рівень доходів та ймовірну ціну продукту, що розробляється.

Встановлено, що хоча розробляєма система і є менш точною в порівнянні з наявними на ринку рішеннями, але через значно нижчу ціну та можливість використання у автомобілях, що були вироблені без широкого спектру датчиків, дана система потенційно має більш широку аудиторію користувачів, що теоретично може призвезти до того, що проект зможе зайняти нішу, яка ще не містить значної конкуренці.

ВИСНОВКИ

В даній магістерській дисертації було досліджено можливість використання сучасних підходів машинного навчання для задачі розпізнаванавання об’єктів в реальному часі, а також використання їх в автомобільній промисловості.

В ході виконання роботи було проведено ознайомлення з системами безпеки відомих автомобільних концернів та методами які вони використовують. Данне дослідження наведено у розділі 1. Також було проаналізовано датчики за допомогою яких отримуються данні для систем розпізнавання, виконано порівняння їх сильних і слабких сторін.

В другому розділі було проаналізовано алгоритми розпізнавання об’єктів, виконано порівняння двоетапних та одноетапних детекторів, виявлено їх слабкі та сильні сторони та но основі аналізу визначено той алгоритм, який використовувався для реалізації цілей встановлених даною магістерською дисертацією.

В третьому розділі було наведено архітектуру обраної системи, проведено її навчання та проаналізовано результати. Загалом було встановлено, що створена система забезпечує досить високі результати в плані швидкодії та повноти розпізнаваних об’єктів, але може мати дещо нижчу точність в складних умовах. Також було підтверджено можливість розпізнавання системою об’єктів в реальному часі, що при доопрацюванні та деякій модифікації інтерфейсу дозволяє її використання в якості стартап проекту, що має досить високий потенціал.

ПЕРЕЛІК ПОСИЛАНЬ

1. Wang, X.Z.; Li, J.; Li, H.J.; Shang, B.X. Obstacle detection based on 3d laser scanner and range image for intelligent vehicle. J. Jilin Univ. (Eng. Technol. Ed.) URL: https://www.mdpi.com/2227-7390/6/10/213/pdf
2. Mercedes-Benz “Intelligent drive” In The NEW E-Class URL: https://www.dsf.my/2014/06/mercedes-benz-intelligent-drive-in-the-new-e-class/
3. Toyota Safety Sense ™ Pre-Collision System (PCS) URL: https://www.cavatoyota.com/blog/watch-video-about-how-the-toyota-pre-collision-system-pcs-works/
4. M. H. Putra, Z. M. Yussof, K. C. Lim, S. I. Salim “Convolutional Neural Network for Person and Car Detection using YOLO Framework”, URL: https://webcache.googleusercontent.com/search?q=cache:DgvmAefA_xAJ:https://core.ac.uk/download/pdf/229273247.pdf+&cd=1&hl=en&ct=clnk&gl=ua
5. Sijia Chen, Yu Wang, Li Huang Runzhou, Ge Yihan Hu, Zhuangzhuang Ding, Jie Liao, Waymo Open Dataset Challenge - 2D Object Detection, arXiv preprint arXiv:2005.15507, Jun. 2020. URL: https://arxiv.org/abs/2006.15507
6. Rohith Gandhi “R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithms” URL: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e
7. Dominguez-Sanchez, A.; Cazorla, M.; Orts-Escolano, S. Pedestrian Movement Direction Recognition Using Convolutional Neural Networks. IEEE Trans. Intell. Transp. Syst. 2017, 18, 3540–3548, doi:10.1109/TITS.2017.2726140.
8. Ren, S.; He, K.; Girshick, R.; Sun, J. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. IEEE Trans. Pattern Anal. Mach. Intell. 2017, 39, doi:10.1109/TPAMI.2016.2577031.
9. Ordóñez, F.J.; Roggen, D. Deep convolutional and LSTM recurrent neural networks for multimodal wearable activity recognition. Sensors 2016, 16, 115, doi:10.3390/s16010115.

10. L. Wang, H. Lu, X. Ruan, and M.-H. Yang, “Deep networks for saliency detection via local estimation and global search,”, URL: https://openaccess.thecvf.com/content_cvpr_2015/papers/Wang_Deep_Networks_for_2015_CVPR_paper.pdf

11. Hiai, Fumio; Lin, Minghua, On an eigenvalue inequality involving the Hadamard product. Linear Algebra and its Applications, Volume 515, 15 February 2017, p 313-320, doi:10.1016/j.laa.2016.11.017

12. S. Ioffe and C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, arXiv preprint arXiv:1502.03167, Feb. 2015. URL: https://arxiv.org/abs/1502.03167

13. G. Howard, Some improvements on deep convolutional neural network based image classification, arXiv preprint arXiv:1312.5402, Dec. 2013. URL: https://arxiv.org/abs/1312.5402

14. Overview of the YOLO Object Detection Algorithm, URL: https://medium.com/@ODSC/overview-of-the-yolo-object-detection-algorithm-7b52a745d3e0#:~:text=YOLO%20is%20a%20clever%20convolutional,and%20probabilities%20for%20each%20region.

15. Abdullah Asım YILMAZ, Mehmet Serdar GÜZEL, İman ASKERBEYLİ, Erkan BOSTANCI, A Vehicle Detection Approach using Deep Learning Methodologies, arXiv preprint arXiv:1804.00429, Apr 2018 URL : https://arxiv.org/abs/1804.00429

ДОДАТОК А ЛІСТИНГ ПРОГРАМИ

```
import logging
import os
import random
import time
from pathlib import Path
from threading import Thread
from warnings import warn

import math
import numpy as np
import torch.distributed as dist
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torch.optim.lr_scheduler as lr_scheduler
import torch.utils.data
import yaml
from torch.cuda import amp
from torch.nn.parallel import DistributedDataParallel as DDP
from torch.utils.tensorboard import SummaryWriter
from tqdm import tqdm

import test  # import test.py to get mAP after each epoch
from models.experimental import attempt_load
from models.yolo import Model
from utils.autoanchor import check_anchors
from utils.datasets import create_dataloader
from utils.general import labels_to_class_weights, increment_path,
labels_to_image_weights, init_seeds, \
    fitness, strip_optimizer, get_latest_run, check_dataset, check_file,
check_git_status, check_img_size, \
    print_mutation, set_logging
from utils.google_utils import attempt_download
from utils.loss import compute_loss
from utils.plots import plot_images, plot_labels, plot_results, plot_evolution
from utils.torch_utils import ModelEMA, select_device, intersect_dicts,
torch_distributed_zero_first

logger = logging.getLogger(__name__)

try:
    import wandb
except ImportError:
    wandb = None
    logger.info("Install Weights & Biases for experiment logging via 'pip install
wandb' (recommended)")


def train(hyp, opt, device, tb_writer=None, wandb=None):
    logger.info(f'Hyperparameters {hyp}')
    save_dir, epochs, batch_size, total_batch_size, weights, rank = \
        Path(opt.save_dir), opt.epochs, opt.batch_size, opt.total_batch_size,
opt.weights, opt.global_rank

    # Directories
    wdir = save_dir / 'weights'
    wdir.mkdir(parents=True, exist_ok=True)  # make dir
    last = wdir / 'last.pt'
    best = wdir / 'best.pt'
    results_file = save_dir / 'results.txt'

    # Save run settings
```

```
    with open(save_dir / 'hyp.yaml', 'w') as f:
        yaml.dump(hyp, f, sort_keys=False)
    with open(save_dir / 'opt.yaml', 'w') as f:
        yaml.dump(vars(opt), f, sort_keys=False)

    # Configure
    plots = not opt.evolve  # create plots
    cuda = device.type != 'cpu'
    init_seeds(2 + rank)
    with open(opt.data) as f:
        data_dict = yaml.load(f, Loader=yaml.FullLoader)  # data dict
    with torch_distributed_zero_first(rank):
        check_dataset(data_dict)  # check
    train_path = data_dict['train']
    test_path = data_dict['val']
    nc, names = (1, ['item']) if opt.single_cls else (int(data_dict['nc']),
data_dict['names'])  # number classes, names
    assert len(names) == nc, '%g names found for nc=%g dataset in %s' % (len(names),
nc, opt.data)  # check

    # Model
    pretrained = weights.endswith('.pt')
    if pretrained:
        with torch_distributed_zero_first(rank):
            attempt_download(weights)  # download if not found locally
        ckpt = torch.load(weights, map_location=device)  # load checkpoint
        if hyp.get('anchors'):
            ckpt['model'].yaml['anchors'] = round(hyp['anchors'])  # force autoanchor
        model = Model(opt.cfg or ckpt['model'].yaml, ch=3, nc=nc).to(device)  # create
        exclude = ['anchor'] if opt.cfg or hyp.get('anchors') else []  # exclude keys
        state_dict = ckpt['model'].float().state_dict()  # to FP32
        state_dict = intersect_dicts(state_dict, model.state_dict(), exclude=exclude)
# intersect
        model.load_state_dict(state_dict, strict=False)  # load
        logger.info('Transferred %g/%g items from %s' % (len(state_dict),
len(model.state_dict()), weights))  # report
    else:
        model = Model(opt.cfg, ch=3, nc=nc).to(device)  # create

    # Freeze
    freeze = []  # parameter names to freeze (full or partial)
    for k, v in model.named_parameters():
        v.requires_grad = True  # train all layers
        if any(x in k for x in freeze):
            print('freezing %s' % k)
            v.requires_grad = False

    # Optimizer
    nbs = 64  # nominal batch size
    accumulate = max(round(nbs / total_batch_size), 1)  # accumulate loss before
optimizing
    hyp['weight_decay'] *= total_batch_size * accumulate / nbs  # scale weight_decay

    pg0, pg1, pg2 = [], [], []  # optimizer parameter groups
    for k, v in model.named_modules():
        if hasattr(v, 'bias') and isinstance(v.bias, nn.Parameter):
            pg2.append(v.bias)  # biases
        if isinstance(v, nn.BatchNorm2d):
            pg0.append(v.weight)  # no decay
        elif hasattr(v, 'weight') and isinstance(v.weight, nn.Parameter):
            pg1.append(v.weight)  # apply decay

    if opt.adam:
        optimizer = optim.Adam(pg0, lr=hyp['lr0'], betas=(hyp['momentum'], 0.999))  #
adjust beta1 to momentum
    else:
        optimizer = optim.SGD(pg0, lr=hyp['lr0'], momentum=hyp['momentum'],
nesterov=True)
```

```
    optimizer.add_param_group({'params': pg1, 'weight_decay': hyp['weight_decay']})  #
add pg1 with weight_decay
    optimizer.add_param_group({'params': pg2})  # add pg2 (biases)
    logger.info('Optimizer groups: %g .bias, %g conv.weight, %g other' % (len(pg2),
len(pg1), len(pg0)))
    del pg0, pg1, pg2

    # Scheduler https://arxiv.org/pdf/1812.01187.pdf
    #
https://pytorch.org/docs/stable/_modules/torch/optim/lr_scheduler.html#OneCycleLR
    lf = lambda x: ((1 + math.cos(x * math.pi / epochs)) / 2) * (1 - hyp['lrf']) +
hyp['lrf']  # cosine
    scheduler = lr_scheduler.LambdaLR(optimizer, lr_lambda=lf)
    # plot_lr_scheduler(optimizer, scheduler, epochs)

    # Logging
    if wandb and wandb.run is None:
        opt.hyp = hyp  # add hyperparameters
        wandb_run = wandb.init(config=opt, resume="allow",
                               project='YOLOv5' if opt.project == 'runs/train' else
Path(opt.project).stem,
                               name=save_dir.stem,
                               id=ckpt.get('wandb_id') if 'ckpt' in locals() else
None)
    loggers = {'wandb': wandb}  # loggers dict

    # Resume
    start_epoch, best_fitness = 0, 0.0
    if pretrained:
        # Optimizer
        if ckpt['optimizer'] is not None:
            optimizer.load_state_dict(ckpt['optimizer'])
            best_fitness = ckpt['best_fitness']

        # Results
        if ckpt.get('training_results') is not None:
            with open(results_file, 'w') as file:
                file.write(ckpt['training_results'])  # write results.txt

        # Epochs
        start_epoch = ckpt['epoch'] + 1
        if opt.resume:
            assert start_epoch > 0, '%s training to %g epochs is finished, nothing to
resume.' % (weights, epochs)
        if epochs < start_epoch:
            logger.info('%s has been trained for %g epochs. Fine-tuning for %g
additional epochs.' %
                        (weights, ckpt['epoch'], epochs))
            epochs += ckpt['epoch']  # finetune additional epochs

        del ckpt, state_dict

    # Image sizes
    gs = int(max(model.stride))  # grid size (max stride)
    imgsz, imgsz_test = [check_img_size(x, gs) for x in opt.img_size]  # verify imgsz
are gs-multiples

    # DP mode
    if cuda and rank == -1 and torch.cuda.device_count() > 1:
        model = torch.nn.DataParallel(model)

    # SyncBatchNorm
    if opt.sync_bn and cuda and rank != -1:
        model = torch.nn.SyncBatchNorm.convert_sync_batchnorm(model).to(device)
        logger.info('Using SyncBatchNorm()')

    # EMA
```

```
    ema = ModelEMA(model) if rank in [-1, 0] else None

    # DDP mode
    if cuda and rank != -1:
        model = DDP(model, device_ids=[opt.local_rank], output_device=opt.local_rank)

    # Trainloader
    dataloader, dataset = create_dataloader(train_path, imgsz, batch_size, gs, opt,
                                            hyp=hyp, augment=True,
cache=opt.cache_images, rect=opt.rect, rank=rank,
                                            world_size=opt.world_size,
workers=opt.workers,
                                            image_weights=opt.image_weights)
    mlc = np.concatenate(dataset.labels, 0)[:, 0].max()  # max label class
    nb = len(dataloader)  # number of batches
    assert mlc < nc, 'Label class %g exceeds nc=%g in %s. Possible class labels are 0-
%g' % (mlc, nc, opt.data, nc - 1)

    # Process 0
    if rank in [-1, 0]:
        ema.updates = start_epoch * nb // accumulate  # set EMA updates
        testloader = create_dataloader(test_path, imgsz_test, total_batch_size, gs,
opt,  # testloader
                                       hyp=hyp, cache=opt.cache_images and not
opt.notest, rect=True,
                                       rank=-1, world_size=opt.world_size,
workers=opt.workers, pad=0.5)[0]

        if not opt.resume:
            labels = np.concatenate(dataset.labels, 0)
            c = torch.tensor(labels[:, 0])  # classes
            # cf = torch.bincount(c.long(), minlength=nc) + 1.  # frequency
            # model._initialize_biases(cf.to(device))
            if plots:
                Thread(target=plot_labels, args=(labels, save_dir, loggers),
daemon=True).start()
                if tb_writer:
                    tb_writer.add_histogram('classes', c, 0)

            # Anchors
            if not opt.noautoanchor:
                check_anchors(dataset, model=model, thr=hyp['anchor_t'], imgsz=imgsz)

    # Model parameters
    hyp['cls'] *= nc / 80.  # scale coco-tuned hyp['cls'] to current dataset
    model.nc = nc  # attach number of classes to model
    model.hyp = hyp  # attach hyperparameters to model
    model.gr = 1.0  # iou loss ratio (obj_loss = 1.0 or iou)
    model.class_weights = labels_to_class_weights(dataset.labels, nc).to(device)  #
attach class weights
    model.names = names

    # Start training
    t0 = time.time()
    nw = max(round(hyp['warmup_epochs'] * nb), 1000)  # number of warmup iterations,
max(3 epochs, 1k iterations)
    # nw = min(nw, (epochs - start_epoch) / 2 * nb)  # limit warmup to < 1/2 of
training
    maps = np.zeros(nc)  # mAP per class
    results = (0, 0, 0, 0, 0, 0, 0)  # P, R, mAP@.5, mAP@.5-.95, val_loss(box, obj,
cls)
    scheduler.last_epoch = start_epoch - 1  # do not move
    scaler = amp.GradScaler(enabled=cuda)
    logger.info('Image sizes %g train, %g test\n'
                'Using %g dataloader workers\nLogging results to %s\n'
                'Starting training for %g epochs...' % (imgsz, imgsz_test,
dataloader.num_workers, save_dir, epochs))
```

```
    for epoch in range(start_epoch, epochs):  # epoch ------------------------------
--------------------------------
        model.train()

        # Update image weights (optional)
        if opt.image_weights:
            # Generate indices
            if rank in [-1, 0]:
                cw = model.class_weights.cpu().numpy() * (1 - maps) ** 2  # class
weights
                iw = labels_to_image_weights(dataset.labels, nc=nc, class_weights=cw)
# image weights
                dataset.indices = random.choices(range(dataset.n), weights=iw,
k=dataset.n)  # rand weighted idx
            # Broadcast if DDP
            if rank != -1:
                indices = (torch.tensor(dataset.indices) if rank == 0 else
torch.zeros(dataset.n)).int()
                dist.broadcast(indices, 0)
                if rank != 0:
                    dataset.indices = indices.cpu().numpy()

        # Update mosaic border
        # b = int(random.uniform(0.25 * imgsz, 0.75 * imgsz + gs) // gs * gs)
        # dataset.mosaic_border = [b - imgsz, -b]  # height, width borders

        mloss = torch.zeros(4, device=device)  # mean losses
        if rank != -1:
            dataloader.sampler.set_epoch(epoch)
        pbar = enumerate(dataloader)
        logger.info(('\n' + '%10s' * 8) % ('Epoch', 'gpu_mem', 'box', 'obj', 'cls',
'total', 'targets', 'img_size'))
        if rank in [-1, 0]:
            pbar = tqdm(pbar, total=nb)  # progress bar
        optimizer.zero_grad()
        for i, (imgs, targets, paths, _) in pbar:  # batch -------------------------
--------------------------------
            ni = i + nb * epoch  # number integrated batches (since train start)
            imgs = imgs.to(device, non_blocking=True).float() / 255.0  # uint8 to
float32, 0-255 to 0.0-1.0

            # Warmup
            if ni <= nw:
                xi = [0, nw]  # x interp
                # model.gr = np.interp(ni, xi, [0.0, 1.0])  # iou loss ratio (obj_loss
= 1.0 or iou)
                accumulate = max(1, np.interp(ni, xi, [1, nbs /
total_batch_size]).round())
                for j, x in enumerate(optimizer.param_groups):
                    # bias lr falls from 0.1 to lr0, all other lrs rise from 0.0 to
lr0
                    x['lr'] = np.interp(ni, xi, [hyp['warmup_bias_lr'] if j == 2 else
0.0, x['initial_lr'] * lf(epoch)])
                    if 'momentum' in x:
                        x['momentum'] = np.interp(ni, xi, [hyp['warmup_momentum'],
hyp['momentum']])

            # Multi-scale
            if opt.multi_scale:
                sz = random.randrange(imgsz * 0.5, imgsz * 1.5 + gs) // gs * gs  #
size
                sf = sz / max(imgs.shape[2:])  # scale factor
                if sf != 1:
                    ns = [math.ceil(x * sf / gs) * gs for x in imgs.shape[2:]]  # new
shape (stretched to gs-multiple)
                    imgs = F.interpolate(imgs, size=ns, mode='bilinear',
align_corners=False)
```

```
            # Forward
            with amp.autocast(enabled=cuda):
                pred = model(imgs)  # forward
                loss, loss_items = compute_loss(pred, targets.to(device), model)  #
loss scaled by batch_size
                if rank != -1:
                    loss *= opt.world_size  # gradient averaged between devices in DDP
mode

            # Backward
            scaler.scale(loss).backward()

            # Optimize
            if ni % accumulate == 0:
                scaler.step(optimizer)  # optimizer.step
                scaler.update()
                optimizer.zero_grad()
                if ema:
                    ema.update(model)

            # Print
            if rank in [-1, 0]:
                mloss = (mloss * i + loss_items) / (i + 1)  # update mean losses
                mem = '%.3gG' % (torch.cuda.memory_reserved() / 1E9 if
torch.cuda.is_available() else 0)  # (GB)
                s = ('%10s' * 2 + '%10.4g' * 6) % (
                    '%g/%g' % (epoch, epochs - 1), mem, *mloss, targets.shape[0],
imgs.shape[-1])
                pbar.set_description(s)

                # Plot
                if plots and ni < 3:
                    f = save_dir / f'train_batch{ni}.jpg'  # filename
                    Thread(target=plot_images, args=(imgs, targets, paths, f),
daemon=True).start()
                    # if tb_writer:
                    #     tb_writer.add_image(f, result, dataformats='HWC',
global_step=epoch)
                    #     tb_writer.add_graph(model, imgs)  # add model to tensorboard
                elif plots and ni == 3 and wandb:
                    wandb.log({"Mosaics": [wandb.Image(str(x), caption=x.name) for x
in save_dir.glob('train*.jpg')]})

            # end batch ------------------------------------------------------------
-------------------------------
        # end epoch ------------------------------------------------------------------
-------------------------------

        # Scheduler
        lr = [x['lr'] for x in optimizer.param_groups]  # for tensorboard
        scheduler.step()

        # DDP process 0 or single-GPU
        if rank in [-1, 0]:
            # mAP
            if ema:
                ema.update_attr(model, include=['yaml', 'nc', 'hyp', 'gr', 'names',
'stride'])
            final_epoch = epoch + 1 == epochs
            if not opt.notest or final_epoch:  # Calculate mAP
                results, maps, times = test.test(opt.data,
                                                 batch_size=total_batch_size,
                                                 imgsz=imgsz_test,
                                                 model=ema.ema,
                                                 single_cls=opt.single_cls,
                                                 dataloader=testloader,
                                                 save_dir=save_dir,
                                                 plots=plots and final_epoch,
```

```
                                                log_imgs=opt.log_imgs if wandb else
0)

            # Write
            with open(results_file, 'a') as f:
                f.write(s + '%10.4g' * 7 % results + '\n')  # P, R, mAP@.5, mAP@.5-
.95, val_loss(box, obj, cls)
            if len(opt.name) and opt.bucket:
                os.system('gsutil cp %s gs://%s/results/results%s.txt' %
(results_file, opt.bucket, opt.name))

            # Log
            tags = ['train/box_loss', 'train/obj_loss', 'train/cls_loss',  # train
loss
                    'metrics/precision', 'metrics/recall', 'metrics/mAP_0.5',
'metrics/mAP_0.5:0.95',
                    'val/box_loss', 'val/obj_loss', 'val/cls_loss',  # val loss
                    'x/lr0', 'x/lr1', 'x/lr2']  # params
            for x, tag in zip(list(mloss[:-1]) + list(results) + lr, tags):
                if tb_writer:
                    tb_writer.add_scalar(tag, x, epoch)  # tensorboard
                if wandb:
                    wandb.log({tag: x})  # W&B

            # Update best mAP
            fi = fitness(np.array(results).reshape(1, -1))  # weighted combination of
[P, R, mAP@.5, mAP@.5-.95]
            if fi > best_fitness:
                best_fitness = fi

            # Save model
            save = (not opt.nosave) or (final_epoch and not opt.evolve)
            if save:
                with open(results_file, 'r') as f:  # create checkpoint
                    ckpt = {'epoch': epoch,
                            'best_fitness': best_fitness,
                            'training_results': f.read(),
                            'model': ema.ema,
                            'optimizer': None if final_epoch else
optimizer.state_dict(),
                            'wandb_id': wandb_run.id if wandb else None}

                # Save last, best and delete
                torch.save(ckpt, last)
                if best_fitness == fi:
                    torch.save(ckpt, best)
                del ckpt
        # end epoch ----------------------------------------------------------------
--------------------------------
    # end training

    if rank in [-1, 0]:
        # Strip optimizers
        for f in [last, best]:
            if f.exists():  # is *.pt
                strip_optimizer(f)  # strip optimizer
                os.system('gsutil cp %s gs://%s/weights' % (f, opt.bucket)) if
opt.bucket else None  # upload

        # Plots
        if plots:
            plot_results(save_dir=save_dir)  # save as results.png
            if wandb:
                files = ['results.png', 'precision_recall_curve.png',
'confusion_matrix.png']
                wandb.log({"Results": [wandb.Image(str(save_dir / f), caption=f) for f
in files
                                       if (save_dir / f).exists()]})
```

```
        logger.info('%g epochs completed in %.3f hours.\n' % (epoch - start_epoch + 1,
(time.time() - t0) / 3600))

        # Test best.pt
        if opt.data.endswith('coco.yaml') and nc == 80:  # if COCO
            results, _, _ = test.test(opt.data,
                                      batch_size=total_batch_size,
                                      imgsz=imgsz_test,
                                      model=attempt_load(best if best.exists() else
last, device).half(),
                                      single_cls=opt.single_cls,
                                      dataloader=testloader,
                                      save_dir=save_dir,
                                      save_json=True,  # use pycocotools
                                      plots=False)

    else:
        dist.destroy_process_group()

    wandb.run.finish() if wandb and wandb.run else None
    torch.cuda.empty_cache()
    return results


if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('--weights', type=str, default='yolov5s.pt', help='initial
weights path')
    parser.add_argument('--cfg', type=str, default='', help='model.yaml path')
    parser.add_argument('--data', type=str, default='data/coco128.yaml',
help='data.yaml path')
    parser.add_argument('--hyp', type=str, default='data/hyp.scratch.yaml',
help='hyperparameters path')
    parser.add_argument('--epochs', type=int, default=300)
    parser.add_argument('--batch-size', type=int, default=16, help='total batch size
for all GPUs')
    parser.add_argument('--img-size', nargs='+', type=int, default=[640, 640],
help='[train, test] image sizes')
    parser.add_argument('--rect', action='store_true', help='rectangular training')
    parser.add_argument('--resume', nargs='?', const=True, default=False, help='resume
most recent training')
    parser.add_argument('--nosave', action='store_true', help='only save final
checkpoint')
    parser.add_argument('--notest', action='store_true', help='only test final epoch')
    parser.add_argument('--noautoanchor', action='store_true', help='disable
autoanchor check')
    parser.add_argument('--evolve', action='store_true', help='evolve
hyperparameters')
    parser.add_argument('--bucket', type=str, default='', help='gsutil bucket')
    parser.add_argument('--cache-images', action='store_true', help='cache images for
faster training')
    parser.add_argument('--image-weights', action='store_true', help='use weighted
image selection for training')
    parser.add_argument('--device', default='', help='cuda device, i.e. 0 or 0,1,2,3
or cpu')
    parser.add_argument('--multi-scale', action='store_true', help='vary img-size +/-
50%%')
    parser.add_argument('--single-cls', action='store_true', help='train as single-
class dataset')
    parser.add_argument('--adam', action='store_true', help='use torch.optim.Adam()
optimizer')
    parser.add_argument('--sync-bn', action='store_true', help='use SyncBatchNorm,
only available in DDP mode')
    parser.add_argument('--local_rank', type=int, default=-1, help='DDP parameter, do
not modify')
    parser.add_argument('--log-imgs', type=int, default=16, help='number of images for
W&B logging, max 100')
```

```
    parser.add_argument('--workers', type=int, default=8, help='maximum number of
dataloader workers')
    parser.add_argument('--project', default='runs/train', help='save to
project/name')
    parser.add_argument('--name', default='exp', help='save to project/name')
    parser.add_argument('--exist-ok', action='store_true', help='existing project/name
ok, do not increment')
    opt = parser.parse_args()

    # Set DDP variables
    opt.total_batch_size = opt.batch_size
    opt.world_size = int(os.environ['WORLD_SIZE']) if 'WORLD_SIZE' in os.environ else
1
    opt.global_rank = int(os.environ['RANK']) if 'RANK' in os.environ else -1
    set_logging(opt.global_rank)
    if opt.global_rank in [-1, 0]:
        check_git_status()

    # Resume
    if opt.resume:  # resume an interrupted run
        ckpt = opt.resume if isinstance(opt.resume, str) else get_latest_run()  #
specified or most recent path
        assert os.path.isfile(ckpt), 'ERROR: --resume checkpoint does not exist'
        with open(Path(ckpt).parent.parent / 'opt.yaml') as f:
            opt = argparse.Namespace(**yaml.load(f, Loader=yaml.FullLoader))  #
replace
        opt.cfg, opt.weights, opt.resume = '', ckpt, True
        logger.info('Resuming training from %s' % ckpt)
    else:
        # opt.hyp = opt.hyp or ('hyp.finetune.yaml' if opt.weights else
'hyp.scratch.yaml')
        opt.data, opt.cfg, opt.hyp = check_file(opt.data), check_file(opt.cfg),
check_file(opt.hyp)  # check files
        assert len(opt.cfg) or len(opt.weights), 'either --cfg or --weights must be
specified'
        opt.img_size.extend([opt.img_size[-1]] * (2 - len(opt.img_size)))  # extend to
2 sizes (train, test)
        opt.name = 'evolve' if opt.evolve else opt.name
        opt.save_dir = increment_path(Path(opt.project) / opt.name,
exist_ok=opt.exist_ok | opt.evolve)  # increment run

    # DDP mode
    device = select_device(opt.device, batch_size=opt.batch_size)
    if opt.local_rank != -1:
        assert torch.cuda.device_count() > opt.local_rank
        torch.cuda.set_device(opt.local_rank)
        device = torch.device('cuda', opt.local_rank)
        dist.init_process_group(backend='nccl', init_method='env://')  # distributed
backend
        assert opt.batch_size % opt.world_size == 0, '--batch-size must be multiple of
CUDA device count'
        opt.batch_size = opt.total_batch_size // opt.world_size

    # Hyperparameters
    with open(opt.hyp) as f:
        hyp = yaml.load(f, Loader=yaml.FullLoader)  # load hyps
        if 'box' not in hyp:
            warn('Compatibility: %s missing "box" which was renamed from "giou" in %s'
%
                 (opt.hyp, 'https://github.com/ultralytics/yolov5/pull/1120'))
            hyp['box'] = hyp.pop('giou')

    # Train
    logger.info(opt)
    if not opt.evolve:
        tb_writer = None  # init loggers
        if opt.global_rank in [-1, 0]:
```

```
            logger.info(f'Start Tensorboard with "tensorboard --logdir {opt.project}",
view at http://localhost:6006/')
            tb_writer = SummaryWriter(opt.save_dir)  # Tensorboard
        train(hyp, opt, device, tb_writer, wandb)

    # Evolve hyperparameters (optional)
    else:
        # Hyperparameter evolution metadata (mutation scale 0-1, lower_limit,
upper_limit)
        meta = {'lr0': (1, 1e-5, 1e-1),  # initial learning rate (SGD=1E-2, Adam=1E-3)
                'lrf': (1, 0.01, 1.0),  # final OneCycleLR learning rate (lr0 * lrf)
                'momentum': (0.3, 0.6, 0.98),  # SGD momentum/Adam beta1
                'weight_decay': (1, 0.0, 0.001),  # optimizer weight decay
                'warmup_epochs': (1, 0.0, 5.0),  # warmup epochs (fractions ok)
                'warmup_momentum': (1, 0.0, 0.95),  # warmup initial momentum
                'warmup_bias_lr': (1, 0.0, 0.2),  # warmup initial bias lr
                'box': (1, 0.02, 0.2),  # box loss gain
                'cls': (1, 0.2, 4.0),  # cls loss gain
                'cls_pw': (1, 0.5, 2.0),  # cls BCELoss positive_weight
                'obj': (1, 0.2, 4.0),  # obj loss gain (scale with pixels)
                'obj_pw': (1, 0.5, 2.0),  # obj BCELoss positive_weight
                'iou_t': (0, 0.1, 0.7),  # IoU training threshold
                'anchor_t': (1, 2.0, 8.0),  # anchor-multiple threshold
                'anchors': (2, 2.0, 10.0),  # anchors per output grid (0 to ignore)
                'fl_gamma': (0, 0.0, 2.0),  # focal loss gamma (efficientDet default
gamma=1.5)
                'hsv_h': (1, 0.0, 0.1),  # image HSV-Hue augmentation (fraction)
                'hsv_s': (1, 0.0, 0.9),  # image HSV-Saturation augmentation
(fraction)
                'hsv_v': (1, 0.0, 0.9),  # image HSV-Value augmentation (fraction)
                'degrees': (1, 0.0, 45.0),  # image rotation (+/- deg)
                'translate': (1, 0.0, 0.9),  # image translation (+/- fraction)
                'scale': (1, 0.0, 0.9),  # image scale (+/- gain)
                'shear': (1, 0.0, 10.0),  # image shear (+/- deg)
                'perspective': (0, 0.0, 0.001),  # image perspective (+/- fraction),
range 0-0.001
                'flipud': (1, 0.0, 1.0),  # image flip up-down (probability)
                'fliplr': (0, 0.0, 1.0),  # image flip left-right (probability)
                'mosaic': (1, 0.0, 1.0),  # image mixup (probability)
                'mixup': (1, 0.0, 1.0)}  # image mixup (probability)

        assert opt.local_rank == -1, 'DDP mode not implemented for --evolve'
        opt.notest, opt.nosave = True, True  # only test/save final epoch
        # ei = [isinstance(x, (int, float)) for x in hyp.values()]  # evolvable
indices
        yaml_file = Path(opt.save_dir) / 'hyp_evolved.yaml'  # save best result here
        if opt.bucket:
            os.system('gsutil cp gs://%s/evolve.txt .' % opt.bucket)  # download
evolve.txt if exists

        for _ in range(300):  # generations to evolve
            if Path('evolve.txt').exists():  # if evolve.txt exists: select best hyps
and mutate
                # Select parent(s)
                parent = 'single'  # parent selection method: 'single' or 'weighted'
                x = np.loadtxt('evolve.txt', ndmin=2)
                n = min(5, len(x))  # number of previous results to consider
                x = x[np.argsort(-fitness(x))][:n]  # top n mutations
                w = fitness(x) - fitness(x).min()  # weights
                if parent == 'single' or len(x) == 1:
                    # x = x[random.randint(0, n - 1)]  # random selection
                    x = x[random.choices(range(n), weights=w)[0]]  # weighted
selection
                elif parent == 'weighted':
                    x = (x * w.reshape(n, 1)).sum(0) / w.sum()  # weighted combination

                # Mutate
                mp, s = 0.8, 0.2  # mutation probability, sigma
```

```
                    npr = np.random
                    npr.seed(int(time.time()))
                    g = np.array([x[0] for x in meta.values()])  # gains 0-1
                    ng = len(meta)
                    v = np.ones(ng)
                    while all(v == 1):  # mutate until a change occurs (prevent
        duplicates)
                        v = (g * (npr.random(ng) < mp) * npr.randn(ng) * npr.random() * s
        + 1).clip(0.3, 3.0)
                    for i, k in enumerate(hyp.keys()):  # plt.hist(v.ravel(), 300)
                        hyp[k] = float(x[i + 7] * v[i])  # mutate

                # Constrain to limits
                for k, v in meta.items():
                    hyp[k] = max(hyp[k], v[1])  # lower limit
                    hyp[k] = min(hyp[k], v[2])  # upper limit
                    hyp[k] = round(hyp[k], 5)  # significant digits

                # Train mutation
                results = train(hyp.copy(), opt, device, wandb=wandb)

                # Write mutation results
                print_mutation(hyp.copy(), results, yaml_file, opt.bucket)

            # Plot results
            plot_evolution(yaml_file)
            print(f'Hyperparameter evolution complete. Best results saved as:
        {yaml_file}\n'
                  f'Command to train a new model with these hyperparameters: $ python
        train.py --hyp {yaml_file}')


    import
    argparse
        import json
        import os
        from pathlib import Path
        from threading import Thread

        import numpy as np
        import torch
        import yaml
        from tqdm import tqdm

        from models.experimental import attempt_load
        from utils.datasets import create_dataloader
        from utils.general import coco80_to_coco91_class, check_dataset, check_file,
        check_img_size, box_iou, \
            non_max_suppression, scale_coords, xyxy2xywh, xywh2xyxy, set_logging,
        increment_path
        from utils.loss import compute_loss
        from utils.metrics import ap_per_class, ConfusionMatrix
        from utils.plots import plot_images, output_to_target, plot_study_txt
        from utils.torch_utils import select_device, time_synchronized


        def test(data,
                 weights=None,
                 batch_size=32,
                 imgsz=640,
                 conf_thres=0.001,
                 iou_thres=0.6,  # for NMS
                 save_json=False,
                 single_cls=False,
                 augment=False,
                 verbose=False,
                 model=None,
                 dataloader=None,
```

```
             save_dir=Path(''),  # for saving images
             save_txt=False,  # for auto-labelling
             save_conf=False,
             plots=True,
             log_imgs=0):  # number of logged images

    # Initialize/load model and set device
    training = model is not None
    if training:  # called by train.py
        device = next(model.parameters()).device  # get model device

    else:  # called directly
        set_logging()
        device = select_device(opt.device, batch_size=batch_size)
        save_txt = opt.save_txt  # save *.txt labels

        # Directories
        save_dir = Path(increment_path(Path(opt.project) / opt.name,
exist_ok=opt.exist_ok))  # increment run
        (save_dir / 'labels' if save_txt else save_dir).mkdir(parents=True,
exist_ok=True)  # make dir

        # Load model
        model = attempt_load(weights, map_location=device)  # load FP32 model
        imgsz = check_img_size(imgsz, s=model.stride.max())  # check img_size

        # Multi-GPU disabled, incompatible with .half()
https://github.com/ultralytics/yolov5/issues/99
        # if device.type != 'cpu' and torch.cuda.device_count() > 1:
        #     model = nn.DataParallel(model)

    # Half
    half = device.type != 'cpu'  # half precision only supported on CUDA
    if half:
        model.half()

    # Configure
    model.eval()
    is_coco = data.endswith('coco.yaml')  # is COCO dataset
    with open(data) as f:
        data = yaml.load(f, Loader=yaml.FullLoader)  # model dict
    check_dataset(data)  # check
    nc = 1 if single_cls else int(data['nc'])  # number of classes
    iouv = torch.linspace(0.5, 0.95, 10).to(device)  # iou vector for mAP@0.5:0.95
    niou = iouv.numel()

    # Logging
    log_imgs, wandb = min(log_imgs, 100), None  # ceil
    try:
        import wandb  # Weights & Biases
    except ImportError:
        log_imgs = 0

    # Dataloader
    if not training:
        img = torch.zeros((1, 3, imgsz, imgsz), device=device)  # init img
        _ = model(img.half() if half else img) if device.type != 'cpu' else None  #
run once
        path = data['test'] if opt.task == 'test' else data['val']  # path to
val/test images
        dataloader = create_dataloader(path, imgsz, batch_size, model.stride.max(),
opt, pad=0.5, rect=True)[0]

    seen = 0
    confusion_matrix = ConfusionMatrix(nc=nc)
    names = {k: v for k, v in enumerate(model.names if hasattr(model, 'names') else
model.module.names)}
    coco91class = coco80_to_coco91_class()
```

```
    s = ('%20s' + '%12s' * 6) % ('Class', 'Images', 'Targets', 'P', 'R', 'mAP@.5',
'mAP@.5:.95')
    p, r, f1, mp, mr, map50, map, t0, t1 = 0., 0., 0., 0., 0., 0., 0., 0., 0.
    loss = torch.zeros(3, device=device)
    jdict, stats, ap, ap_class, wandb_images = [], [], [], [], []
    for batch_i, (img, targets, paths, shapes) in enumerate(tqdm(dataloader,
desc=s)):
        img = img.to(device, non_blocking=True)
        img = img.half() if half else img.float()  # uint8 to fp16/32
        img /= 255.0  # 0 - 255 to 0.0 - 1.0
        targets = targets.to(device)
        nb, _, height, width = img.shape  # batch size, channels, height, width

        with torch.no_grad():
            # Run model
            t = time_synchronized()
            inf_out, train_out = model(img, augment=augment)  # inference and
training outputs
            t0 += time_synchronized() - t

            # Compute loss
            if training:
                loss += compute_loss([x.float() for x in train_out], targets,
model)[1][:3]  # box, obj, cls

            # Run NMS
            targets[:, 2:] *= torch.Tensor([width, height, width,
height]).to(device)  # to pixels
            lb = [targets[targets[:, 0] == i, 1:] for i in range(nb)] if save_txt
else []  # for autolabelling
            t = time_synchronized()
            output = non_max_suppression(inf_out, conf_thres=conf_thres,
iou_thres=iou_thres, labels=lb)
            t1 += time_synchronized() - t

        # Statistics per image
        for si, pred in enumerate(output):
            labels = targets[targets[:, 0] == si, 1:]
            nl = len(labels)
            tcls = labels[:, 0].tolist() if nl else []  # target class
            path = Path(paths[si])
            seen += 1

            if len(pred) == 0:
                if nl:
                    stats.append((torch.zeros(0, niou, dtype=torch.bool),
torch.Tensor(), torch.Tensor(), tcls))
                continue

            # Predictions
            predn = pred.clone()
            scale_coords(img[si].shape[1:], predn[:, :4], shapes[si][0],
shapes[si][1])  # native-space pred

            # Append to text file
            if save_txt:
                gn = torch.tensor(shapes[si][0])[[1, 0, 1, 0]]  # normalization
gain whwh
                for *xyxy, conf, cls in predn.tolist():
                    xywh = (xyxy2xywh(torch.tensor(xyxy).view(1, 4)) / gn).view(-
1).tolist()  # normalized xywh
                    line = (cls, *xywh, conf) if save_conf else (cls, *xywh)  #
label format
                    with open(save_dir / 'labels' / (path.stem + '.txt'), 'a') as
f:
                        f.write(('%g ' * len(line)).rstrip() % line + '\n')

            # W&B logging
```

```
            if plots and len(wandb_images) < log_imgs:
                box_data = [{"position": {"minX": xyxy[0], "minY": xyxy[1], "maxX":
xyxy[2], "maxY": xyxy[3]},
                             "class_id": int(cls),
                             "box_caption": "%s %.3f" % (names[cls], conf),
                             "scores": {"class_score": conf},
                             "domain": "pixel"} for *xyxy, conf, cls in
pred.tolist()]
                boxes = {"predictions": {"box_data": box_data, "class_labels":
names}}  # inference-space
                wandb_images.append(wandb.Image(img[si], boxes=boxes,
caption=path.name))

            # Append to pycocotools JSON dictionary
            if save_json:
                # [{"image_id": 42, "category_id": 18, "bbox": [258.15, 41.29,
348.26, 243.78], "score": 0.236}, ...
                image_id = int(path.stem) if path.stem.isnumeric() else path.stem
                box = xyxy2xywh(predn[:, :4])  # xywh
                box[:, :2] -= box[:, 2:] / 2  # xy center to top-left corner
                for p, b in zip(pred.tolist(), box.tolist()):
                    jdict.append({'image_id': image_id,
                                  'category_id': coco91class[int(p[5])] if is_coco
else int(p[5]),
                                  'bbox': [round(x, 3) for x in b],
                                  'score': round(p[4], 5)})

            # Assign all predictions as incorrect
            correct = torch.zeros(pred.shape[0], niou, dtype=torch.bool,
device=device)
            if nl:
                detected = []  # target indices
                tcls_tensor = labels[:, 0]

                # target boxes
                tbox = xywh2xyxy(labels[:, 1:5])
                scale_coords(img[si].shape[1:], tbox, shapes[si][0], shapes[si][1])
# native-space labels
                if plots:
                    confusion_matrix.process_batch(pred, torch.cat((labels[:, 0:1],
tbox), 1))

                # Per target class
                for cls in torch.unique(tcls_tensor):
                    ti = (cls == tcls_tensor).nonzero(as_tuple=False).view(-1)  #
prediction indices
                    pi = (cls == pred[:, 5]).nonzero(as_tuple=False).view(-1)  #
target indices

                    # Search for detections
                    if pi.shape[0]:
                        # Prediction to target ious
                        ious, i = box_iou(predn[pi, :4], tbox[ti]).max(1)  # best
ious, indices

                        # Append detections
                        detected_set = set()
                        for j in (ious > iouv[0]).nonzero(as_tuple=False):
                            d = ti[i[j]]  # detected target
                            if d.item() not in detected_set:
                                detected_set.add(d.item())
                                detected.append(d)
                                correct[pi[j]] = ious[j] > iouv  # iou_thres is 1xn
                                if len(detected) == nl:  # all targets already
located in image
                                    break

            # Append statistics (correct, conf, pcls, tcls)
```

```
            stats.append((correct.cpu(), pred[:, 4].cpu(), pred[:, 5].cpu(), tcls))

        # Plot images
        if plots and batch_i < 3:
            f = save_dir / f'test_batch{batch_i}_labels.jpg'  # labels
            Thread(target=plot_images, args=(img, targets, paths, f, names),
daemon=True).start()
            f = save_dir / f'test_batch{batch_i}_pred.jpg'  # predictions
            Thread(target=plot_images, args=(img, output_to_target(output), paths,
f, names), daemon=True).start()

    # Compute statistics
    stats = [np.concatenate(x, 0) for x in zip(*stats)]  # to numpy
    if len(stats) and stats[0].any():
        p, r, ap, f1, ap_class = ap_per_class(*stats, plot=plots,
save_dir=save_dir, names=names)
        p, r, ap50, ap = p[:, 0], r[:, 0], ap[:, 0], ap.mean(1)  # [P, R, AP@0.5,
AP@0.5:0.95]
        mp, mr, map50, map = p.mean(), r.mean(), ap50.mean(), ap.mean()
        nt = np.bincount(stats[3].astype(np.int64), minlength=nc)  # number of
targets per class
    else:
        nt = torch.zeros(1)

    # Print results
    pf = '%20s' + '%12.3g' * 6  # print format
    print(pf % ('all', seen, nt.sum(), mp, mr, map50, map))

    # Print results per class
    if verbose and nc > 1 and len(stats):
        for i, c in enumerate(ap_class):
            print(pf % (names[c], seen, nt[c], p[i], r[i], ap50[i], ap[i]))

    # Print speeds
    t = tuple(x / seen * 1E3 for x in (t0, t1, t0 + t1)) + (imgsz, imgsz,
batch_size)  # tuple
    if not training:
        print('Speed: %.1f/%.1f/%.1f ms inference/NMS/total per %gx%g image at
batch-size %g' % t)

    # Plots
    if plots:
        confusion_matrix.plot(save_dir=save_dir, names=list(names.values()))
        if wandb and wandb.run:
            wandb.log({"Images": wandb_images})
            wandb.log({"Validation": [wandb.Image(str(f), caption=f.name) for f in
sorted(save_dir.glob('test*.jpg'))]})

    # Save JSON
    if save_json and len(jdict):
        w = Path(weights[0] if isinstance(weights, list) else weights).stem if
weights is not None else ''  # weights
        anno_json = '../coco/annotations/instances_val2017.json'  # annotations
json
        pred_json = str(save_dir / f"{w}_predictions.json")  # predictions json
        print('\nEvaluating pycocotools mAP... saving %s...' % pred_json)
        with open(pred_json, 'w') as f:
            json.dump(jdict, f)

        try:  #
https://github.com/cocodataset/cocoapi/blob/master/PythonAPI/pycocoEvalDemo.ipynb
            from pycocotools.coco import COCO
            from pycocotools.cocoeval import COCOeval

            anno = COCO(anno_json)  # init annotations api
            pred = anno.loadRes(pred_json)  # init predictions api
            eval = COCOeval(anno, pred, 'bbox')
            if is_coco:
```

```
                eval.params.imgIds = [int(Path(x).stem) for x in
dataloader.dataset.img_files]  # image IDs to evaluate
            eval.evaluate()
            eval.accumulate()
            eval.summarize()
            map, map50 = eval.stats[:2]  # update results (mAP@0.5:0.95, mAP@0.5)
        except Exception as e:
            print(f'pycocotools unable to run: {e}')

    # Return results
    if not training:
        s = f"\n{len(list(save_dir.glob('labels/*.txt')))} labels saved to
{save_dir / 'labels'}" if save_txt else ''
        print(f"Results saved to {save_dir}{s}")
    model.float()  # for training
    maps = np.zeros(nc) + map
    for i, c in enumerate(ap_class):
        maps[c] = ap[i]
    return (mp, mr, map50, map, *(loss.cpu() / len(dataloader)).tolist()), maps, t


if __name__ == '__main__':
    parser = argparse.ArgumentParser(prog='test.py')
    parser.add_argument('--weights', nargs='+', type=str, default='yolov5s.pt',
help='model.pt path(s)')
    parser.add_argument('--data', type=str, default='data/coco128.yaml',
help='*.data path')
    parser.add_argument('--batch-size', type=int, default=32, help='size of each
image batch')
    parser.add_argument('--img-size', type=int, default=640, help='inference size
(pixels)')
    parser.add_argument('--conf-thres', type=float, default=0.001, help='object
confidence threshold')
    parser.add_argument('--iou-thres', type=float, default=0.6, help='IOU threshold
for NMS')
    parser.add_argument('--task', default='val', help="'val', 'test', 'study'")
    parser.add_argument('--device', default='', help='cuda device, i.e. 0 or
0,1,2,3 or cpu')
    parser.add_argument('--single-cls', action='store_true', help='treat as single-
class dataset')
    parser.add_argument('--augment', action='store_true', help='augmented
inference')
    parser.add_argument('--verbose', action='store_true', help='report mAP by
class')
    parser.add_argument('--save-txt', action='store_true', help='save results to
*.txt')
    parser.add_argument('--save-conf', action='store_true', help='save confidences
in --save-txt labels')
    parser.add_argument('--save-json', action='store_true', help='save a cocoapi-
compatible JSON results file')
    parser.add_argument('--project', default='runs/test', help='save to
project/name')
    parser.add_argument('--name', default='exp', help='save to project/name')
    parser.add_argument('--exist-ok', action='store_true', help='existing
project/name ok, do not increment')
    opt = parser.parse_args()
    opt.save_json |= opt.data.endswith('coco.yaml')
    opt.data = check_file(opt.data)  # check file
    print(opt)

    if opt.task in ['val', 'test']:  # run normally
        test(opt.data,
             opt.weights,
             opt.batch_size,
             opt.img_size,
             opt.conf_thres,
             opt.iou_thres,
             opt.save_json,
```

```
                opt.single_cls,
                opt.augment,
                opt.verbose,
                save_txt=opt.save_txt,
                save_conf=opt.save_conf,
                )

    elif opt.task == 'study':  # run over a range of settings and save/plot
        for weights in ['yolov5s.pt', 'yolov5m.pt', 'yolov5l.pt', 'yolov5x.pt']:
            f = 'study_%s_%s.txt' % (Path(opt.data).stem, Path(weights).stem)  #
filename to save to
            x = list(range(320, 800, 64))  # x axis
            y = []  # y axis
            for i in x:  # img-size
                print('\nRunning %s point %s...' % (f, i))
                r, _, t = test(opt.data, weights, opt.batch_size, i,
opt.conf_thres, opt.iou_thres, opt.save_json,
                               plots=False)
                y.append(r + t)  # results and times
            np.savetxt(f, y, fmt='%10.4g')  # save
        os.system('zip -r study.zip study_*.txt')
        plot_study_txt(f, x)  # plot
```